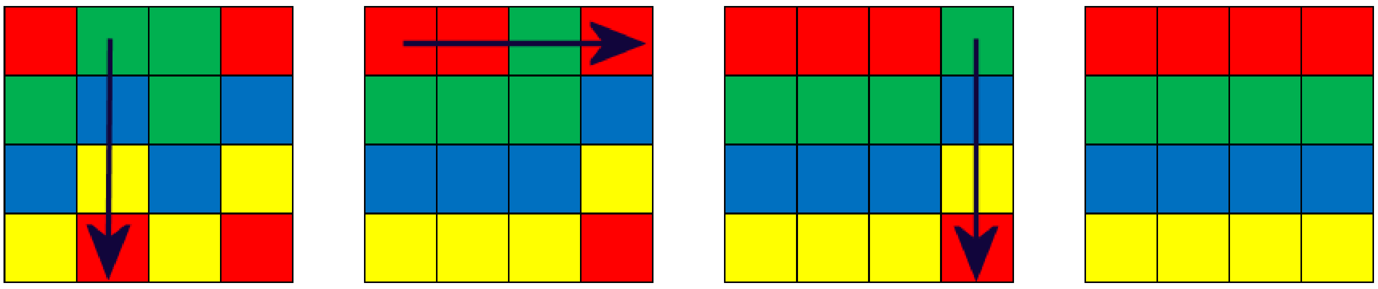
Problem A. Another Rubik's Puzzle?

Input file:	*standard input*
Output file:	*standard output*
Time limit:	8 seconds
Memory limit:	512 mebibytes

You are given a puzzle that can be represented as a 4×4 grid of colored cells. The solved puzzle contains 4 monochromatic rows, in this order: red, green, blue, yellow. Although we will analyze this puzzle using its 2D representation, it is actually a 3D puzzle! Imagine that the grid is stretched over a torus (in other words, top edge is connected to the bottom one and left edge is connected to the right one).



For each move you are allowed to either move one row left or right, or one column up or down. The fact that the outer edges are connected means that if a cell is "pushed out" of the grid, it will reappear on the other side of the grid.

Given a description of a state of this puzzle, what is the minimum number of moves you need to solve it? Note that all possible puzzle configurations are solvable in less than 13 moves.

Input

Input file contains exactly 4 lines, containing 4 characters each, each character being either "R", "G", "B" or "Y". The input will describe a valid state of the puzzle.

Output

Output the minimum number of moves needed to solve the given puzzle.

Examples

standard input	standard output
RGGR GBGB BYBY YRYR	3
RRRR GBGG GYBB BYYY	4

Problem B. Being Solarty Systematic

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 mebibytes

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 21st 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 $n \le 100$ is the number of planetoids, and n_x, n_y and n_z are the dimensions of the space the planetoids reside in, where $n_x, n_y, n_z \le 1000$.

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 $1 \le m \le 100$, $0 \le x < n_x$, $0 \le y < n_y$, $0 \le z < n_z$, and $-1000 \le v_x, v_y, v_z \le 1000$. 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" ($0 \le i < p$), 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.

Examples

standard input	standard output

`2 8 8 8`
`12 4 1 4 5 3 -2`
`10 1 2 1 8 -6 1`
	`1`
	`P0: 22 1 4 2 6 -1 0`

`2 10 20 30`
`10 1 0 0 2 0 0`
`15 2 0 0 4 0 0`
	`2`
	`P0: 15 2 0 0 4 0 0`
	`P1: 10 1 0 0 2 0 0`

Problem C. Cyber-ZOO

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

While keepers of Cyber-ZOO weren't watching, his N robots have developed a life of their own and spread throughout a town. Each of town's N intersections (numbered $0, \ldots, N-1$) contains exactly one robot. On each intersection i, there is exactly one red signpost pointing to an intersection, $r_i \neq i$, and exactly one green signpost pointing to an intersection $g_i \neq i$. When zookeeper presses the red button on his remote control, each robot will move to the intersection indicated by the red signpost (robots at intersection i move to r_i). When he presses the green button, each robot will move to the intersection indicated by the green signpost (robots at intersection i move to g_i). Write a program that determines whether zookeeper can make the robots all meet at the same intersection at the same time via some sequence of commands on their remote control.

Input

The first line of input contains a single decimal integer P, $(1 \leq P \leq 500)$, which is the number of data sets that follow. Each data set should be processed identically and independently. Each data set consists three lines of input as follows:

- The first line contains the data set number, K, followed by a single integer N $(1 \leq N \leq 500)$ which is the number of intersections.
- The second line contains N space-separated integers $r_0, \ldots, r_{N-1}$ $(0 \leq r_i \leq N-1$ and $r_i \neq i)$.
- The third line contains N space separated integers $g_0, \ldots, g_{N-1}$ $(0 \leq g_i \leq N-1$ and $g_i \neq i)$.

On some intersections, both signposts might point the same way (i.e. $r_i = g_i$).

Output

For each data set there is one line of output. The single output line consists of the number of data set and string "`YES`" if zookeeper can make all robots meet or "`NO` otherwise.

Example

standard input	standard output

2
1 4
1 2 3 0
3 0 1 0
2 4
1 2 3 0
2 2 1 2
	1 NO
	2 YES

Note

Note: For the second case, the button press sequence "`GREEN`", "`RED`", "`RED`", "`GREEN`" makes all robots meet at intersection 2.

Problem D. Diversity of Tree

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

Given a tree with n vertices, we randomly choose k vertices of it. Then we can induced a subtree which is the smallest connected subtree of the original tree containing those k vertices.

Each vertex have a color, for a subtree we induced, we look at its diameter — path a-b (if there are many diameters, pick the one with the smallest a, and then the smallest b). Define *diversity* of a this tree as number of distinct colors which are on the diameter.

What is the expected divercity of this tree?

Input

The first line contains an integer T, $(1 \leq T \leq 20)$, denoting the number of the test cases. For each test case, the first line contains two integers n and k $(1 \leq n, k \leq 300)$. Each of the next $n-1$ lines contain two integers a and b, denoting there is an edge between vertices a and b. The next line contains n integers c_i separated by a single space, denote each vertex's color in the order from 1 to n $(1 \leq c_i \leq 10^9)$.

Output

Print expected diversity of a given tree with absolute or relative error 10^{-6} or less

Examples

standard input	standard output

1
20 8
2 1
3 1
4 1
5 4
6 3
7 2
8 4
9 5
10 5
11 10
12 11
13 10
14 11
15 12
16 12
17 14
18 13
19 18
20 17
5 6 2 1 2 4 7 3 1 3 5 4 1 7 2 6 1 2 1
5
	5.7866158609

Problem E. Expectation

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

There are n points in the plane, you pick a random point in $[0, X] \times [0, Y]$, after that you output the squared distance from this point to second nearest to it between one of those n points. What is the expectation of your output?

Input

The first line contains an integer T ($1 \le T \le 5$), denoting the number of the test cases. For each test case, the first line contains 3 integers n, X, Y ($2 \le n \le 100$, $0 \le X, Y \le 100$, $-200 \le x_i, y_i \le 200$. The next n lines, each contains 2 integers x_i, y_i, denote that there is a point x_i,y_i.

Output

For each test case, output the answer in one line. The absolute or relative error less than 10^{-6} will be accepted.

Example

standard input	standard output

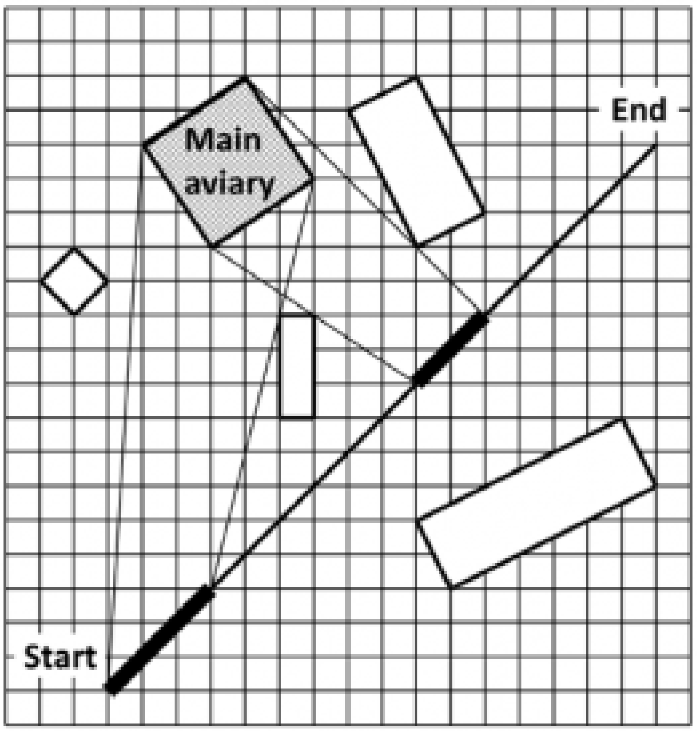
1
2 10 10
0 0
10 10
	100.0000000000

Problem F. Flight Cage

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

An aviary or a flight cage is a big cage for birds. An usual ZOO aviary typically measures tens of meters in diameter. In the aviaries, the birds can fly around and live in conditions imitating the conditions in the wild as closely as possible. At least in theory. There is one main big and spectacular aviary in the ZOO and some other less important ones.

The ZOO is planning to build a short straight electric train track to help visitors to move easily from one part of the ZOO to another. It has to be decided which of the free areas of the ZOO will the track run through. The director had noticed during his trips to other ZOOs that the visitors are more happy when they can take more photos of important ZOO structures. Now he wants to measure the quality of the planned railway by this parameter. The most important structure in the vicinity of the track will be the main aviary. The director worries that the main aviary might be obscured by the less important aviaries along the track and the visitors might be less happy. Help the director to assess the quality of the planned track.



You are given the coordinates of all aviaries. Also, you are given the coordinates of the start and the end of the planned railway track. Find the total length of the segments on the track from which the main aviary is visible and is not obscured, even not partially, by any other aviary. We suppose that the visitors can look out from the train in any direction.

Input

There are no more than 210 test cases. Each case occupies several lines. The first line contains number N ($1 \leq N \leq 100$) of the aviaries. Next line contains the coordinates of the planned railway track in the format x_1, y_1, x_2, y_2, where $[x_1, y_1]$ and $[x_2, y_2]$ are the coordinates of the start and the end of the track. The track is considered to be infinitely thin in this representation. Next, there are N lines specifying the aviaries, each aviary is represented as a rectangle with nonzero area. Each of these lines specifies the coordinates of an aviary in the form x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, where $[x_1, y_1]$, $[x_2, y_2]$, $[x_3, y_3]$, and $[x_4, y_4]$ are the coordinates of the aviary corners. The corners are presented in clockwise or anti-clockwise order. The main aviary is listed first. All coordinates are integers, their absolute value is less than 10^4. You may assume that no aviary intersects or touches the track or another aviary. There is no blank line between consecutive test cases. The input is terminated by a line with one zero.

Output

For each test case print on a separate line the total length L of all segments of the planned track from which the main aviary is visible and it is not obscured, even not partially, by any other aviary. Your answer should not differ from the correct answer by more than 10^{-4}.

Example

standard input	standard output

5
3 1 17 15
6 14 4 17 7 19 9 16
2 12 1 13 2 14 3 13
8 9 8 12 9 12 9 9
12 14 10 18 12 19 14 15
12 6 18 9 19 7 13 4
1
0 0 0 2
4 -1 4 1 5 1 5 -1
2
0 0 0 1
4 0 4 2 5 2 5 0
2 0 3 0 3 -1 2 -1
2
0 0 0 1
4 0 4 2 5 2 5 0
2 0 3 0 3 1 2 1
0
	7.07105
	2.00
	1.00
	0.00

Note

As shown in the third Sample Input, the main aviary is not considered obscured if only its corners/edges are hidden.

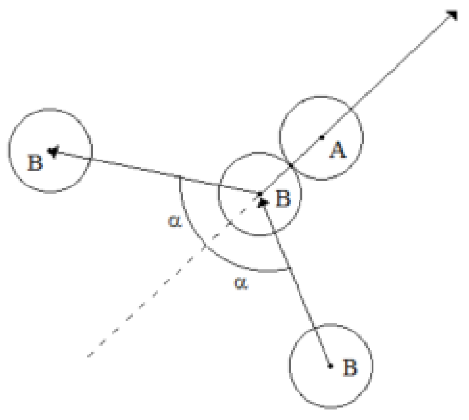
Problem G. Game Physics

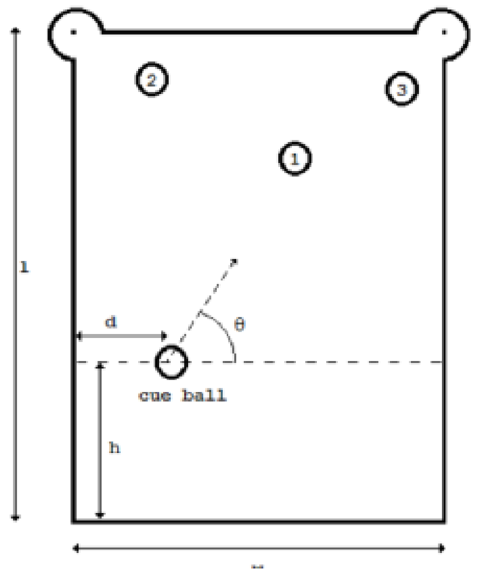
Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 mebibytes

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 (left figure). 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.





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 (right figure). 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 α 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 and 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 α to shoot the ball, or the word "`impossible`" if the trick shot cannot be done. Output α in degrees, and round both d and α to the nearest hundredth.

Example

standard input	standard output
20 30 2 10 20 2 24 18 28 10	12.74 127.83
20 30 2 15 20 2 24 18 28 10	impossible

Problem H. Herrings

Input file:	*standard input*
Output file:	*standard output*
Time limit:	7 seconds
Memory limit:	512 mebibytes

Zookeper Willy is feeding herrings today. He is feeding them to seals, as they are their preferred food. There are three separate pools in which the seals live. The ZOO is a modern institution and it demands their employees to keep track of feeding habits of animals. There is a touchscreen installed at the seals pools and Willy has to enter the number of herrings which he is going to deposit into each of the three pools. Unfortunately, the screen is not working properly - in particular, it is impossible to enter the digit "3".

Willy called the chief marine mammals zookeper and asked for help.

"That is OK", said the chief, "just distribute the herrings in such a way that the number of herrings which go into each pool does not contain the digit 3".

"But there is a lower limit L on the number of herrings which have to be put into each pool", reacted Willy, "I might not be able to find a suitable division".

"You will be able to find a suitable division", assured him the chief, "considering the numbers of herrings in the bucket, there should be zillions of possible divisions".

"Well, exactly how many?" — wondered Willy for himself.

You will be given the total number N of herrings which are to be deposited into the seals pools and the lower limit L on the number of herrings in each of the pools. Find out in how many ways might these N herrings be placed into the pools in such a way that the number of herrings in each pool does not contain digit 3 in its decimal representation. In this problem, we do not distinguish between individual herrings as they are all more or less of the same size and nutrition value. We do distinguish between the pools, though, because they are populated by different groups of seals. Also, we suppose that no herring can be divided into pieces.

Input

There are no more than 500 test cases. Each case consists of a single line containing two integers N and L ($1 \leq N \leq 10^{10000}$, $1 \leq L \leq N/3$) separated by space and representing the number of herrings in the bucket and the lower limit on the number of herrings which have to be deposited in each of the pools. The input is terminated by a line with two zeros.

Output

For each test case print on a separate line the number of possible divisions of the herrings into the three given pools. Express the result modulo 12345647.

Example

standard input	standard output

`3 1`
`4 1`
`7 2`
`99999 1`
`0 0`
	`1`
	`3`
	`0`
	`9521331`

Problem I. Integer Sequence

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	512 mebibytes

There are n integers $a_1, a_2, \dots a_n$ on a line, and queries of two types are coming:

1. Replace each number in a segment $[l, r]$ with an integer x.
2. Replace each $a_i > x$ in a segment $[l, r]$ with $gcd(a_i, x)$ and keep all $a_i \le x$ untouched.

Given an initial sequence and an queries, you should output the final sequence.

Input

The first line of the input contains an integer T ($1 \le T \le 5$), denoting the number of the test cases. For each test case, the first line contains a integer n ($1 \le n \le 10^5$).

The next line contains n integers $a_1, a_2, \dots, a_n$ separated by a single space ($1 \le a_i \le 2^{31} - 1$).

The next line contains an integer Q, denoting the number of the queries ($1 \le Q \le 10^5$). Each of the next Q lines denotes one query and contains 4 integers t ($1 \le t \le 2$) — type of query, l ($1 \le l \le n$) — leftmost position in a segment, r ($l \le r \le n$) — rightmost position in a segment, and x ($1 \le x \le 2^{31} - 1$) — parameter of a query.

Output

For each test case, output a line with n integers separated by a single space, representing the final sequence.

Examples

standard input	standard output

1
10
16807 282475249 1622650073 984943658
1144108930 470211272 101027544
1457850878 1458777923 2007237709
10
1 3 6 74243042
2 4 8 16531729
1 3 4 1474833169
2 1 8 1131570933
2 7 9 1505795335
2 3 7 101929267
1 4 10 1624379149
2 2 8 2110010672
2 6 7 156091745
1 2 5 937186357
	16807 937186357 937186357 937186357
	937186357 1 1 1624379149 1624379149
	1624379149

Problem J. Johnny's Quest

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

The teacher wrote n integers $a_1, a_2, \ldots, a_n$ are written on the board and asked little Johnny to choose two sets S $(a_{s_1}, a_{s_2}, \ldots, a_{s_k})$ and T $(a_{t_1}, a_{t_2}, \ldots, a_{t_m})$, such as:

- Each element in S should be at the left of every element in T.($s_i < t_j$ for all i, j). S and T shouldn't be empty.
- Bitwise XOR of all elements in S is equal to the bitwise AND all elements in T.

How many ways exists for Johnny to choose such two sets? You should output the result modulo $10^9 + 7$.

Input

The first line of the input contains an integer T $(1 \leq T \leq 20)$, denoting the number of the test cases. For each test case, the first line contains a integer n $(1 \leq n \leq 10^3)$. The next line contains n integers $a_1, a_2, \ldots, a_n$ $(0 \leq a_i < 1024)$ which are separated by a single space.

Output

For each test case, output the result in one line.

Examples

standard input	standard output

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

Problem K. KenKen You Do It?

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

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.

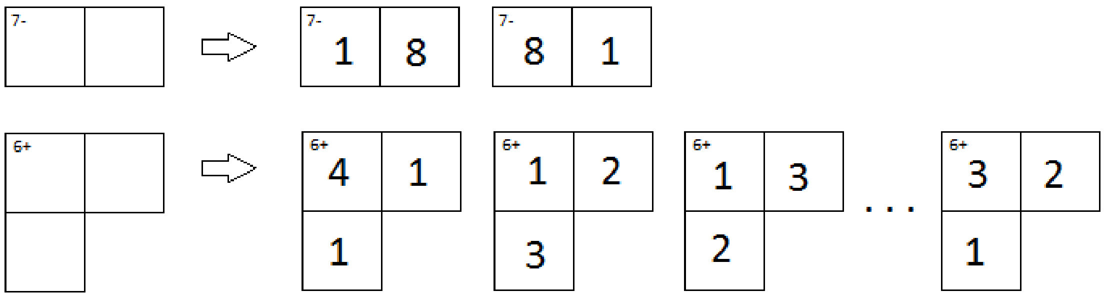


Figure K.1

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 $4 \le n \le 9$, $2 \le m \le 10$, $0 < t$ and $1 \le r, c \le n$.

Output

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

Examples

standard input	standard output

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

`9 2 7 -`
`1 1 1 2`
	`4`

`8 3 6 +`
`5 2 6 2 5 1`
	`7`

Problem L. Leprechaun Hunt

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 mebibytes

In Irish mythology, a Leprechaun is a small sprite who stores all his treasure in a hidden pot of gold at the end of the rainbow. If someone is able to catch the Leprechaun, he must give that person his pot of gold. In this problem, we explore the difficulty of capturing a Leprechaun.

We model a search with V villagers trying to catch a single Leprechaun as a game on a simple undirected graph having $N \geq 1 + V$ nodes. To begin the game, the villagers position themselves at a subset of V distinct nodes. After that, the Leprechaun chooses a remaining node as a starting position. In each round of the game that follows, one villager moves from his or her current node to an adjacent node that is unoccupied by another villager. If that node has the Leprechaun, the villagers win the pot of gold. Otherwise, the Leprechaun now has the option of either staying at his current node, or moving to an adjacent, unoccupied node. Given a specific graph, and a fixed number of villagers, we are interested in the minimum number of turns the villagers need to capture the most clever of Leprechauns.

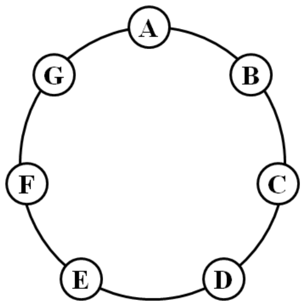


Figure 1

As examples, consider the two figures below. For the graph in Figure 1, a single villager can never capture a Leprechaun, as the Leprechaun can easily stay away from the villager. However, two villagers can capture the Leprechaun after at most 2 turns. For example, the villagers might begin at nodes A and D, in which case a clever Leprechaun will start at node F. But after the villager at A moves to G the villagers can capture the Leprechaun on their second turn, no matter whether the Leprechaun moves to E or remains at F.

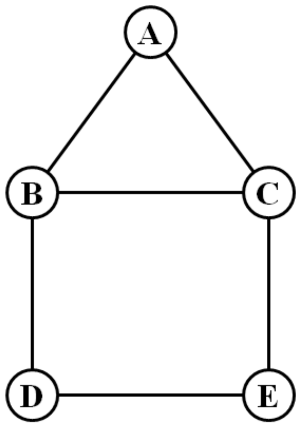


Figure 2

For the graph in Figure 2, a single villager is unable to catch a clever Leprechaun. To see why this is the case, we describe a possible strategy of the Leprechaun, which is to always stay within the square made by $BCDE$, and opposite of the villager if the villager is in that square. If the villager were ever to go to

A, the Leprechaun can remain still. In contrast, two villagers are able to capture the Leprechaun on their first move by picking initial positions such as B and E.

Input

Each test begins with a line containing three integers: V, N and E. The value of V denotes the number of villagers such that $1 \le V \le 7$. The number of nodes N in the graph will satisfy $1 + V \le N \le 15$. The value $1 \le E \le 45$ designates the number of edges in the graph. Following the initial line of parameters will be one or more lines describing the edges of the graph, with up to 15 edges per line. Nodes of the graph are implicitly denoted with the first N uppercase letters (‘`A`’, ‘`B`’, ‘`C`’), and edges are explicitly denoted as two-character strings; for example the string “`AC`” denotes an edge connecting nodes A and C to each other. The E edges will be distinct, each edge connects two distinct nodes, and any node will have at most 6 incident edges. A line with the single value 0 designates the end of the input. Total number of testcases in this problem does not exceed 36.

Output

For each test case, output a line, prefaced with the case number as shown in the example output below, followed by the minimum number of moves that the villagers need to guarantee capture of the Leprechaun, or the word “`NEVER`” if the villagers are unable to capture the Leprechaun.

Example

standard input

```
1 7 7
AB BC CD DE EF FG GA
2 7 7
AB BC CD DE EF FG GA
1 5 6
AB AC BC BD DE EC
2 5 6
AB AC BC BD DE EC
2 10 15
AB BC CD DE EA AF BG CH DI EJ FH HJ
JG GI IF
3 10 15
AB BC CD DE EA AF BG CH DI EJ FH HJ
JG GI IF
3 14 10
AB BC CD EF FG GH IJ JK LM MN
4 14 10
AB BC CD EF FG GH IJ JK LM MN
0
```

standard output

```
CASE 1: NEVER
CASE 2: 2
CASE 3: NEVER
CASE 4: 1
CASE 5: NEVER
CASE 6: 1
CASE 7: NEVER
CASE 8: 2
```

Problem M. Mosaic

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

In this problem, we consider a special class of tile mosaics, as exemplified in Figure G.1. Each such mosaic is built on a rectangular grid with a white background. Within each cell of the grid is either a square black tile, a triangular black tile in one of the four orientations shown in Figure G.2, or nothing, in which case the grid cell remains white. Furthermore, each mosaic is designed so that any shape that remains white is rectangular (possibly rotated).

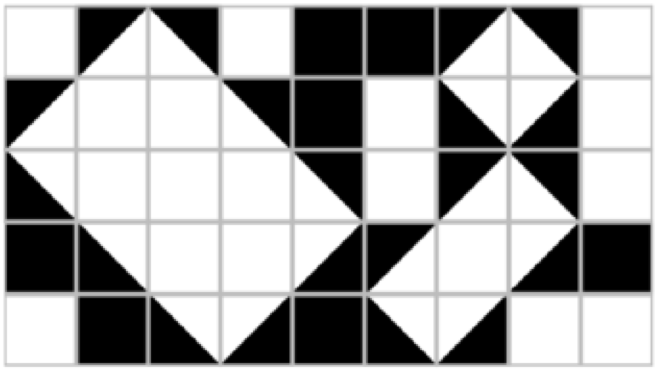


Figure G.1: An example of a mosaic

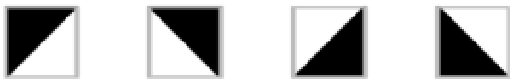


Figure G.2: Orientations for triangular tiles

To install such a mosaic, an artist starts by placing all the black squares. Remaining black triangles will later be added by assistants in order to complete the pattern. To ensure that the assistants complete the mosaic as envisioned, the artist marks some of the black tiles with a numeric label that indicates the number of black triangles that share an edge with that square. (Black tiles without a label may have any number of neighboring triangles.) The artists provides enough labels to ensure a unique design.

For example, Figure G.3 shows a starting configuration that uniquely defines the mosaic shown in Figure G.1. Given such a starting configuration, you are to determine the number of triangles needed to complete the mosaic.

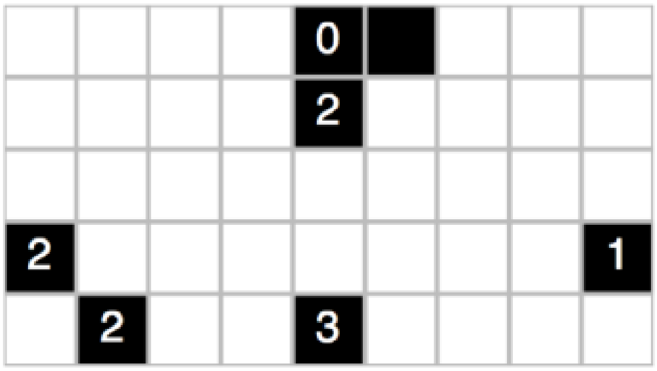


Figure G.3: A starting configuration that uniquely defines the mosaic from Figure G.1

Input

The input consists of a single test case. The first line contains two integers, $1 \le W \le 24$ and $1 \le H \le 18$, that designate the width and height of the mosaic, respectively. Following that are H additional lines, each with W characters. The characters '0', '1', '2', '3', and '4' designate black squares with the indicated constraint on the number of neighboring triangles, and the character '*' designates a black square without such a constraint. All remaining locations will be designated with a '.' character and must either be left empty or covered with a single black triangle. Inputs have been chosen so that they define a valid and unique mosaic.

Output

Display the number of triangles used in the mosaic.

Examples

standard input	standard output

`9 5`
`....0*...`
`....2....`
`.........`
`2.......1`
`.2..3....`
	`20`

`5 5`
`2...*`
`.....`
`.....`
`.....`
`*...0`
	`14`

`18 10`
`*1....*2.....**2..`
`2.............3...`
`...4..............`
`....4..*.....*....`
`2*...*3.....3.....`
`.....*.....**...3*`
`....3.....3..3....`
`..............4...`
`...1.............0`
`..1*2.....2*....1*`
	`92`