Problem A. Gholam's Simple Game

The floor of Gholam's bedroom is tiled with white and yellow tiles. Sometimes when he is bored, he stands on one of the tiles and starts to walk along the row he is standing on. He first decides on a number *n* and starts to walk *n* steps. If he reaches the wall, he turns back and continues to walk in the opposite direction. He continues until he takes *n* steps. Note that turning back besides a wall does not count as a step. He counts how many yellow tiles he steps on.

For example, the figure on the right shows a row in the floor. The colors of the tiles are shown with the characters 'Y' and 'W' for yellow and white tiles respectively. If he starts at tile 3 facing to the right, and decides to take 7 steps, he finally stops at tile 2. During this walk, he steps 3 times on yellow tiles.

1	2	3	4	5	6
Y	W	W	Y	W	Y

Input (Standard Input)

The input contains *T* test cases. The first line of input has only the integer *T*. Each test case contains two lines. The first line contains two integers m ($3 \le m \le 100$) and n ($1 \le n \le 1000$), which is the number of steps Gholam takes. The second line contains *m* integers describing the tiles in the row and is in the following format:

$a_1\ a_2\ \ldots\ a_m$

Each a_i is either 0, 1, 2, or 3. If $a_i = 0$, then a_i has a yellow tile, and $a_i > 0$ indicates that a_i has a white tile. If $a_i = 2$, then Gholam is starting from the tile a_i, facing to the right, and if $a_i = 3$, then he is starting from the tile a_i, facing to the left. The numbers are separated by space characters. You may assume that exactly one of the numbers is 2 or 3. Note that it is implied that Gholam always starts from a white tile.

Output (Standard Output)

For each test case, write a single line in the output having a single number which is the number of times Gholam steps on a yellow tile.

Sample Input and Output

The first test case corresponds to the example given in the problem description.

Standard Input	Standard Output

2
6 7
0 1 2 0 1 0
5 3
0 3 1 0 0
	3
	1

Problem B. Arbiter Login

Finally, the budget for enhancing Arbiter web site (which is an online judge) is allocated and your job is to improve the functionality of its login page. The new login page should help users on two common login mistakes:

1. When the “caps lock” key is ON, each lowercase letter is typed as uppercase and vice versa.
2. When “num lock” key is OFF, numeric characters (digits ‘0’ through ‘9’) are not typed.

Given a username and a password in the login page, the system compares the entered password with the original one (for that username). If they were the same, the login step is successfully finished and the user is redirected to the appropriate page. Otherwise, the login is unsuccessful. But, some extra hints should be given to the user if the entered password was related to the original one in the following ways:

1. If the two passwords were case-insensitively the same, but the case of all their English letters were completely opposite, then the “caps lock” key might be ON by mistake.
2. If the entered password was the same as the original one after removing the numeric characters of the original password, then the “num lock” key might be OFF by mistake.
3. The two cases above might happen together.

Given both the entered password and its original one, you have to write a program which prints the appropriate message.

Input (Standard Input)

The input contains n test cases. The first line of input has only the integer n. Each of the next n lines is a test case having two strings of at least one and at most 10 alphanumeric characters. The first string is the original password, and the second one is the password entered by the user in the login page.

Output (Standard Output)

Write the result of the ith test case, on the ith line of output. As you see in the sample output, you should write the test case number and then, one of the following messages:

- `Login successful.`
- `Wrong password. Please, check your caps lock key.`
- `Wrong password. Please, check your num lock key.`
- `Wrong password. Please, check your caps lock and num lock keys.`
- `Wrong password.`

Note that you must write the output exactly as indicated. Pay attention to character cases and blanks.

Sample Input and Output

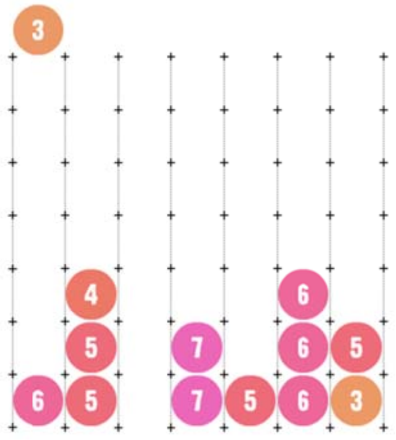
Standard Input

```
6
aaBBccDD aaBBccDD
aaBBccDD aaBBccDD9
aaBBccDD aaBBCCDD
aaBBccDD AAbbCCdd
a4B3c2D1 aBcD
a4B3c2D1 AbCd
```

Standard Output

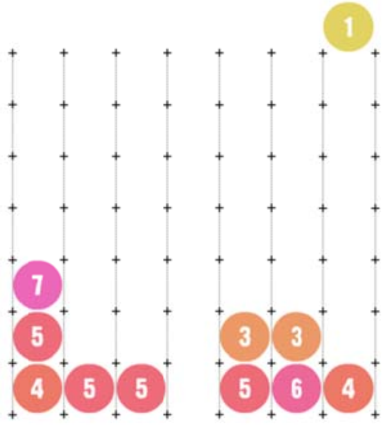
```
Case 1: Login successful.
Case 2: Wrong password.
Case 3: Wrong password.
Case 4: Wrong password. Please, check your caps lock key.
Case 5: Wrong password. Please, check your num lock key.
Case 6: Wrong password. Please, check your caps lock and num lock keys.
```

Problem C. Drop7



Drop7 is a single-player game played on a 7×7 vertical grid. Initially, the grid is empty. In each turn, you will be given a disc on which a number between 1 and 7 is written and you have to drop it into one of the 7 columns from the above. It falls down the 7th row of the selected column, until it reaches above an already occupied cell or touches the ground in the first (bottom) row. For example, in Figure 1, dropping the disc into the first (leftmost) column causes it to stand in the second row (above 6) and dropping it into the third column causes it to hit the ground and to stay in the first row.

After dropping the given disc in a turn, each disc having a number equal to the size of its group-row or group-column will disappear (we will shortly define group-row and group-column). We know that all the discs that are subject to disappear will fade simultaneously. After that, all discs that the disc below them has been disappeared will fall down as far as possible. Then, the disappearance condition will be checked again on all the available discs again and if some disc number matches their group-row/column size, the loop continues. Otherwise, the next turn comes and a new disc will be dropped.



The group-column of a disc is the number of discs that are already placed in the same column, including itself. For example, if we drop the disc 3 into the first column (as in Figure 1), its group-column will have the size of 2 and nothing happens. But, if we drop it into the 4th column from the left, the disc will disappear (because the group-column consists of three discs, 7, 7, and 3). Interestingly, if we drop this disc over the 7th (rightmost) column, not only the disc itself will disappear, but also the other 3 will disappear at the same time! More interestingly, if we drop this disc on the 2nd column, two discs will be removed in two steps – first the disc 4 will disappear as the column size (after putting 3 over it) will be 4. Then, after removal of 4, disc 3 will fall down over the two fives and then disappears by the same rule of matching number with group-column size.

Group-row size of a disc, however, is the number of discs connectedly in the same row of the disc. For example, in Figure 2, if we drop disc 1 into the second column, the group-row size of it, besides the 5 in the first column, would be two. But, if we drop it into the last column, the group-row size of both of 3's will be come 3 and they both will disappear at the same time. Then, the disc 1 will have group-row size of 1 and will disappear by itself!

Starting with an empty board, you will be given some disc-drop actions. You just have to simulate the game and write down the final state of the board in the output.

Input (Standard Input)

The input consists of several test cases. Each test case begins with a line containing the number n ($1 \le n \le 1000$) which is the number of discs dropped. In the following next n lines, n pairs of v_i c_i will be given, which states that at turn i, a disc with number v_i is dropped into the column c_i. It is guaranteed that $1 \le v_i, c_i \le 7$. The last line of the input contains a single zero.

Output (Standard Output)

For each test case of the input, write seven lines of length seven, indicating the disc numbers in the grid. Put a hash sign (#) for empty cells. See the sample output for clarification. If during the game, a disc is dropped into a column that already contains 7 discs, just write “Game Over!” for the output of the test case. Write a blank link after the output of each test case.

Sample Input and Output

Standard Input	Standard Output

6
2 2
3 2
6 2
4 3
1 1
6 7
11
5 3
5 3
2 3
2 3
2 3
3 3
4 3
5 3
6 3
7 3
1 3
0
	#######
	#######
	#######
	#######
	#######
	#######
	#64###6
	
	Game Over!

Problem D. Seven Segment Counter

A Seven Segment Display is generally a form for representing decimals numbers 0 to 9, using exactly 7 segments. You may have already seen such displays on price-list boards, like in exchange offices or fuel stations. Using three of these displays together, with a proper underlying electronic circuit, one can create a three digit display to show number between 0 and 999 (inclusive), without leading zero. Our man, Shelby, has already done this! Going further, he has even made it to work as a time counter – As each second passes, the number in display advances by one and when reaching 999, the next number is 0 (i.e. it works as a cyclic counter).

Shelby is a technician in a company. There is one of those three-digit seven segment displays (working as a counter) hanging on Shelby's office wall. Shelby is asked to write the displaying number of the counter on a piece of paper, everyday at 15:00:00 that is the time he leaves the company. It has some security reasons and is requested by the big boss!

Unfortunately, yesterday Shelby forgot to write the number and he is too anxious now! As a matter of luck he has found some images taken by various surveillance cameras taken at 15:00:00 or later. But, the problem is that, the state (either on or off) of some of the 21 segments (in three seven segments) are not visible in the photos clearly!

Given the photos, each containing information about those 21 segments (either visible and on, or visible and off, or not-visible in this certain photo), and the time the photos is taken, you are asked to help Shelby to find out the number of the counter at 15:00:00.

Input (Standard Input)

The input consists of several test cases, each containing some photos (at least one and at most 20) . Each photo is shown in 6 lines. The first line has a single non-negative integer, less than or equal to 61200, indicating the number of seconds passed since 15:00:00. Then, in the following 5 lines, the photo is drawn. If all the 21 segments of the photo are visible and on (value equals to 888), then the following characters will make the five lines containing space, dash (minus), and vertical bar characters:

```
 -  -  -
| || || |
 -  -  -
| || || |
 -  -  -
```

Otherwise, for any segment (of these 21) that is visible but is off, a dot character (`.`) is written instead of `-` or `|`. For those segments which are not visible, an asterisk character (`*`) is put. You can assume that the digits 0 to 9 are supposed to be displayed like this in a clear photo:

	-	
|		|
|		|
	-	

		|
		|

	-	
		|
	-	
|		
	-	

	-	
		|
	-	
		|
	-	

|		|
	-	
		|

	-	
|		
	-	
		|
	-	

	-	
|		
	-	
|		|
	-	

	-	
		|
		|

	-	
|		|
	-	
|		|
	-	

	-	
|		|
	-	
		|
	-	

The test cases in the input are separated by a line containing a single # sign. The last line of the input contains a single $ sign.

There are not any leading spaces but trailing spaces might occur. Thus, every line has at least (and not exactly) 9 characters in length. Remember the numbers in counter does not contain leading zeros. For example, the value 70 is shown as 7 on the second digit from the left and 0 on the third digit from the left. The leftmost digit is all-off (it is not 070).

Output(Standard Output)

For every test case, if it is possible to find out the value of counter at 15:00:00, then find it and write it in the output. Otherwise, write a question mark (?) first, then a space followed by the number of different possible values the counter may have had at 15:00:00. Separate the question mark and the number by one blank character. If it is an impossible case and the photos are contradicting with each other, you should write "`? 0`" in the output.

Sample Input and Output

Standard Input

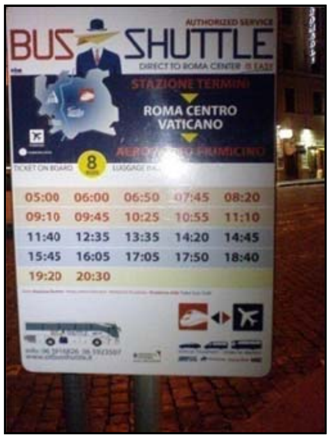
```
1
 -  -  -
| || || |
 -  -  -
| || || |
 -  -  -
#
0
 *  *  -
* ** *| |
 *  *  -
* ** *. |
 *  *  -
1
 .  .  *
. |. |* *
 .  .  *
. |. |* *
 .  .  *
#
6120
 -  -  -
| || || *
 -  *  -
* || || |
 -  -  -
$
```

Standard Output

```
887
109
? 8
```

Problem E. Bus Shuttle

Pasha is a passenger in Rome, Italy. He is in the train station terminal right now (named *Stazione Termini* in Italian) and wants to go to the airport of Rome (named *Aeroporto Fiumicino*) as soon as possible.



05:00	06:00	06:50	07:45	08:20
09:10	09:45	10:25	10:55	11:10
11:40	12:35	13:35	14:20	14:45
15:45	16:05	17:05	17:50	18:40
19:20	20:30			

Just in front of the exit door of Termini, he found a bus station for Bus Shuttle service which takes the passengers right to airport for only 8 Euros per passenger, luggage included! “Sounds economic and fast!”, said Pasha with himself and decided to wait for the bus.

Under the sign for Bus Shuttle, there was a time table in which only leaving time of buses was written. As it is around 7:00 AM in Rome now, he has to wait until 7:45 AM for the first bus to come.

Pasha is curious to know the travel time between *Stazione Termini* and *Aeroporto Fiumicino*. He asks a passenger in the bus station but they do not know it or their English are too weak to talk to Pasha! However, a passenger tells Pasha:

> “I remember, it was yesterday at 13:15, that I was in a bus shuttle to *Termini* and saw another bus shuttle (going to *Aeroporto*) just passed us, in the opposite direction. But, I can’t remember when I got sit in the bus and when we arrived to *Aeroporto*. All I remember is a cross with another shuttle at 13:15”.

Not enough information, but still useful. A few minutes later, some other passengers gave the same type of information to Pasha – exactly a meet at a certain time with a shuttle bus coming in opposite direction while they were in a bus.

And finally Pasha found a brochure of the shuttle service, in which not only the time table of “Termini to Aeroporto” was written, but also the time table of “Aeroporto to Termini” could be found (i.e. the same table in the figure, but in the Aeroporto station and for buses that are leaving Aeroporto to Termini).

Having these two time tables and some meeting-times (like 13:15 in above example), Pasha asks for your assistance to discover the travel time between Termini and Aeroporto, in minutes.

You may assume that all buses are moving with constant speed and without stop in the middle of the way. We know that the travel time is less than 24 hours. You can also assume each bus goes to parking after arriving to its destination and all the buses are leaving in a single day i.e. if there are n leaving times written on Termini station and m leaving times are on Aeroporto station, then assume only $n + m$ buses exist in Rome, which have $n \times m$ possible meeting times.

Input (Standard Input)

The input consists of several test cases. Each test case begins with a line containing two numbers n (the number of buses going from Termini to Aeroporto) and m (the number of buses going from Aeroporto to Termini) in the day of our story. It is guaranteed that $0 < n, m \leq 100$.

In the second line of each test case, n distinct and space-separated times are given between 00:00 and 23:59 (inclusive) with leading zero when minutes or hours are single digit. They are the times a buses leaves Termini to Aeroporto. The third line contains m distinct times, in the same format, for the buses leaving Aeroporto to Termini. You can assume either all of these $m + n$ times are even, or all of them are odd (in minutes).

The fourth line contains an integer $1 \leq k \leq 100$, the number of passengers who have given “meeting time while was in bus” to Pasha. These k times are written in the fifth line in the same format for time as above and separated by space.
Hint: Although all buses are leaving in the same day but they may meet the day after! For example, if the travel time is 8 hours, then a bus leaving at 22:00 from A to T will meet the bus leaving at 23:00 from T to A at 06:30 next day. This way, a report of 06:30 may belong the morning of the same day (e.g. a bus left at 05:00 met the bus left at 06:00 of opposite direction and the travel time is 2 hours, or the prior case). Remember that a meet can occur even in stations!

For example, a bus may leave at 21:05 and another bus from the opposite direction arrives exactly at the same time. The last line of the input contains to zero numbers.

Output (Standard Output)

For each test case, if the travel time can be found and is unique, write it in a single line. If no travel time can satisfy the claims of the passengers write "`il bugiardo passeggeri!`" (without quotes) for this case and proceed to the the next line. Otherwise, if there are $c > 1$ possible travel times (in range 00:01 to 23:59, of course) write "c `scelte`" where c is substituted with its value. Note that the travel times cannot have fractions of minutes.

Sample Input and Output

Standard Input	Standard Output
2 2 05:05 06:05 12:09 11:09 1 17:00 2 1 05:00 06:00 13:00 2 14:00 15:00 2 2 05:05 06:05 12:09 11:09 2 17:00 18:00 0 0	3 scelte il bugiardo passeggeri! 17:46

Problem F. Conditional Statements

Bonjol is given a new task in his company. He is given a piece of code written in Pascal programming language for controlling a set of lights, and his task is to refactor the code and minimize its size while keeping the program logic exactly the same as before. The code is made up of several conditional statements. Each conditional statement is a line of the form

`if` ⟨*variable*⟩ ⟨*comparison-operator*⟩ ⟨*comparison-value*⟩ **`then turnOn(`** ⟨*light-number*⟩ **`);`**

where

- ⟨*variable*⟩ is a string of at most 255 lowercase English letters which is the name of a Pascal integer variable;
- ⟨*comparison-operator*⟩ is either "<", ">", or "=";
- ⟨*comparison-value*⟩ is a 32-bit integer constant to which the ⟨*variable*⟩ is compared;
- ⟨*light-number*⟩ is another 32-bit integer constant which shows the number of the light which should be turned on if the condition "⟨*variable*⟩ ⟨*comparison-operator*⟩ ⟨*comparison-value*⟩" holds. (Nothing happens if the light is already turned on.)

Here is an example of such a code:

The only code modification which Bonjol is allowed is to delete a complete line. He wants to delete as many lines as possible such that the modified program remains completely equivalent to its original version. You are to help him and calculate the maximum number of lines which could be deleted.

Input (Standard Input)

The input contains several test cases. Each test case starts with a line containing an integer *n* ($1 \le n \le 500$) which is the number of lines in the code. Each of the next *n* lines has a conditional statement. In each conditional statement, there is a single space after "`if`", the ⟨*variable*⟩, the ⟨*comparison-operator*⟩, the ⟨*comparison-value*⟩, "`then`", "`turnOn(`", and the ⟨*light-number*⟩. The input terminates with a line containing "`0`" which should not be processed as a test case.

Output (Standard Output)

Write the result of the *i*th test case, on the *i*th line of output. You should just write one integer indicating the maximum number of lines that Bonjol can delete.

Sample Input and Output

Standard Input	Standard Output
`7`	`3`
`if aa < 4 then turnOn( 3 );`	`1`
`if aa > 7 then turnOn( 3 );`	`2`
`if aa = 7 then turnOn( 3 );`	
`if aa = -1 then turnOn( 3 );`	
`if aa < 7 then turnOn( 3 );`	
`if b = 6 then turnOn( 4 );`	
`if b = 6 then turnOn( 3 );`	
`4`	
`if pq = 6 then turnOn( 7 );`	
`if pq = 6 then turnOn( 7 );`	
`if pq = 6 then turnOn( 1 );`	
`if pq = 2 then turnOn( 1 );`	
`4`	
`if pq = 6 then turnOn( 8 );`	
`if pq = 6 then turnOn( 8 );`	
`if pq = 6 then turnOn( 9 );`	
`if pq < 8 then turnOn( 9 );`	
`0`	

Problem G. Genome Evolution

Xi, a developmental biologist is working on developmental distances of chromosomes. A chromosome, in the Xi's simplistic view, is a permutation from n genes numbered 1 to n. Xi is working on an evolutionary distance metric between two chromosomes. In Xi's theory of evolution any subset of genes lying together in both chromosomes is a positive witness for chromosomes to be similar.

A positive witness is a pair of sequences of the same length *A* and *A'*, where *A* is a consecutive subsequence of the first chromosome, *A'* is a consecutive subsequence of the second chromosome, and *A* is a permutation of *A'*. The goal is to count the number of positive witnesses of two given chromosomes that have a length greater than one.

Input (Standard Input)

There are several test cases in the input. Each test case starts with a line containing the number of genes ($2 \le n \le 3000$). The next two lines contain the two chromosomes, each as a list of positive integers. The input terminates with a line containing "`0`" which should not be processed as a test case.

Output(Standard Output)

For each test case, output a single line containing the number of positive witnesses for two chromosomes to be similar.

Sample Input and Output

Standard Input	Standard Output

4
3 2 1 4
1 2 4 3
5
3 2 1 5 4
3 2 1 5 4
0
	3
	10

Problem H. World Cup Nominations

Every four years, the world is enthusiastically celebrating the Football World Cup event. Hosting such a prestigious event is an honor for a country whichpours in billions of dollars. That is why there are many competitions among volunteer countries and even some shadow-bribing to get the vote to be the host of the World Cup.

Voting procedure in the selection ceremony is yet another important part of the host. Several countries $\{C_1, C_2, ..., C_n\}$ stand as the candidates for the host. Repeatedly, two countries are selected randomly, and are put to vote. The losing country is eliminated and the process is continued until only one country remains, which is the winner.

We have run a poll beforehand to predict the possible outcome of the voting. For a pairs of countries C_i and C_j, we know which one is the winner if the two countries are put in vote against each other. There is no possibility of ties in the voting.

The problem is, having the result of polls, determine the set of countries for which there is a possibility to win. In other words, rule out the countries for which we are sure there is no chance of becoming the host.

Input (Standard Input)

There are several test cases in the input. The first line of each test case contains a single integer n ($0 < n \leq 1000$), the number of countries. Then, there are n lines, describing the poll results. The ith line ($1 \leq i \leq n$) is a string of zeros and ones of length $n - i$. The jth character in the string is one if C_i wins over C_j, and zero otherwise. Note that the last line of each test case is an empty line. The last line of the input contains a single 0.

Output (Standard Output)

For each test case, there is a single line in the output containing the number of countries that have a chance for becoming the World Cup host, according to the given poll results.

Sample Input and Output

Standard Input	Standard Output
3 10 0 0	1

Problem I. Killbots

Killbots is a simple game of evading killer robots. Who created the robots and why they have been programmed to destroy, no one knows. All that is known is that the robots are numerous and their sole objective is to destroy everything on their way. Fortunately, their creator has focused on quantity rather than quality and as a result the robots are severely lacking in intelligence. Killbots is a turn-based game, played on a rectangular *grid*. Each cell can contain one of the following:

Hero (@) – The hero is the unfortunate soul stuck in a room with a bunch of killer robots. In the original Killbots game, the player controls the hero by moving it around the grid. But in this problem, it is assumed that the hero stays at his place and does not move at all. Instead, it can destroy any robot trying to come to its cell.

Robot (+) – Robots are mechanical thugs desiring only to crush the hero with their metallic girth. They have never heard of the "Three Laws of Robotics" and would probably crush anyone who tried to explain them. Each robot will take a single step toward the hero on every turn. If the robot is in the same row or the same column as the hero, it will step directly towards to robot. Otherwise, it will take the diagonal step that makes it as closer to the hero.

Fastbot (#) – ***Fastbots*** are a much speedier version of the basic robot. For each move a robot makes, they take two steps. Fortunately, their increased mobility was not paired with increased intelligence; they often only succeed in destroying themselves twice as quickly as their slower brethren.

Junkheap (*) – When two or more robots or fastbots collide, they are destroyed and the resulting debris produces a junkheap. Robots or fastbots blindly ignore any junkheaps in their path and are destroyed upon crashing into them.

The following figure is an example of how the game works. The cell E5 is where the hero stands. The cells containing junkheap are marked with *. Robots and fastbots are shown with '+' and '#' respectively. If we start from the grid on the left, the robots at B5 and E4 take direct steps directly towards the hero (to the right and down respectively). The latter is destroyed by the hero. The one at B8 takes an up-right step to C7. The robot at H6 takes an up-left step to G5 where it crashes into junkheap and is destroyed. The robot at F7 toes to E6 (where it will be destroyed in the next step by the fastbot initially at G8. Note that all robots and fastbots move simultaneously, so at the same time F7 moves to E6, the fastbot at G8 moves to F7, without collision. After a turn that all robots and fastbots move, the fastbots move another step while the robots stay. So, the fastbot at F7 moves to E6 crashing into the robot staying there. They are both destroyed leaving junkheap at E6. The game continues until all robots and fastbots are destroyed either by the hero, or crashing into other robots or junkheap.

	A	B	C	D	E	F	G	H
1				*				
2								#
3						*		
4					+			
5		+			@		*	
6								+
7		*				+		
8		+					#	

	A	B	C	D	E	F	G	H
1				*				
2								
3						*		
4						#		
5			+		@		*	
6					*			
7		*	+					
8								

The goal of this problem is count the number of robots and fastbots destroyed by the hero.

Input (Standard Input)

There are multiple test-cases in the input, each specifying a game grid. Each test case contains several lines of the same length. Each line describes a row of the grid, and each character of the line is a cell of the grid. Empty cells are marked by '.', and non-empty cells are marked by the special characters described in the problem description. After each test-case, there is a line in the input containing a single '$' character. You may assume there is one and only one cell in the input containing '@'. Each side of the grid has at least one and at most 1000 cells.

Output (Standard Output)

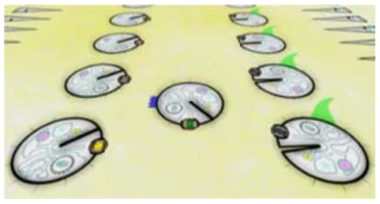
For each test case, write a single line containing the number of fastbots and robots destroyed by the hero when the game ends (after all robots and fastbots are destroyed).

Sample Input and Output

Standard Input	Standard Output

`...*....`
`.......#`
`.....*..`
`....+...`
`.+..@.*.`
`.......+`
`.*...+..`
`.+....#.`
`$`
`...`
`..@`
`.*.`
`.+.`
`$`
	`4`
	`1`

Problem J. Indisputable Right



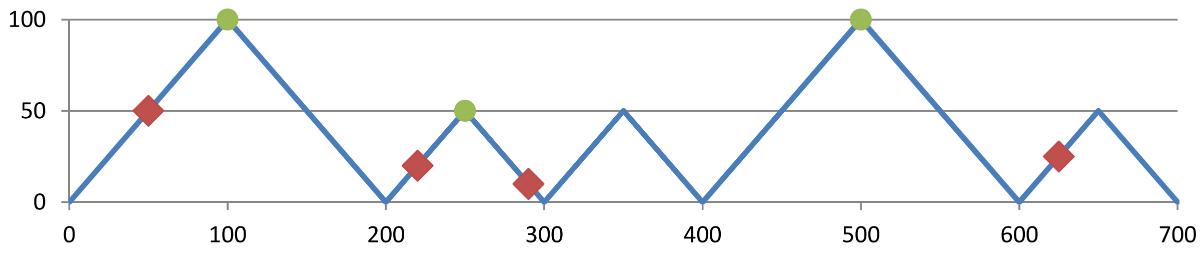
Flatland is a 2-dimensional world in which some creatures have an odd miserable life. Despite the many handicaps, the poor creatures have recently achieved the wireless technology in Flatland. So, the priest circles (governors of Flatland who kill the apostles of 3D as soon as they see one) created the national motto "*Wireless technology is our indisputable right!*" and organized a wireless spreading committee for connecting all important points of the Flatland by wireless antennas.

Everything went well until the committee reached the mountainous region of Flatland. This region can be modeled with a horizontal *base line* covered by a contiguous series of mountains. Each mountain is a right-angled isosceles triangle with its base on the *base line* and its right angle as the peak. Currently, there are only two types of mountains in this region: those with a height of 50, and those with a height of 100. You can see a sample of the mountainous region in the figure below.

The wireless spreading committee first installed their antennas on the points they had already planned for the mountainous region. But in the end, they found out that a pair of antennas could directly communicate only if the line segment connecting the two antennas had no crossing intersection with any obstacles — specifically the mountains here. Note that two antennas *can* communicate if *only the border* of obstacles is on their connecting line segment.

Since removing a wireless antenna in Flatland is nowadays as a deadly sin as "chromatization", the only way to connect the set of already installed antennas is to install some extra antennas on the appropriate points of the mountains! Wireless antennas can generally relay the messages and connect the ones which cannot communicate directly. In order to reduce the expenses of installing the extra antennas, the wireless spreading committee has hired you to find the minimum number of antennas needed to install.

For example, consider the 4 diamond-shaped points in the figure below as the place of the wireless antennas already installed on the mountains. Now the whole set becomes connected if you install 3 new antennas in the places specified by the circle-shaped points.



Input (Standard Input)

There are multiple test cases in the input. Each test case starts with a line containing the integer n $(1 \le n \le 500)$, the number of mountains, followed by the integer m $(1 \le m \le 1000)$, the number of already installed antennas. The second line contains n integers $h_1, h_2, \dots, h_n$ $(h_i \in \{50, 100\})$, respectively representing the height of the mountains on the *base line* from left to right. The third (last) line of the test case contains m integers specifying the position of the antennas already installed on the mountains. The position of each antenna is specified by its X coordinate, assuming the left-most point of the left-most mountain being the origin of the coordinate system. You yourself can calculate the Y coordinate of an antenna as it is put on the boundaries of a mountain.

The input terminates with a line of the form "`0 0`" which should not be processed as a test case.

Output (Standard Output)

For each test case, write a single line containing the minimum number of extra wireless antennas needed for making the given set of antennas connected.

Sample Input and Output

This test case is related to the configuration illustrated in the above figure.

Standard Input	Standard Output
`5 4`	`3`
`100 50 50 100 50`	
`50 220 290 625`	
`0 0`	

Problem K. The Mayo Empire

In the history of Asia there is an ancient empire called Mayo which was one of the most powerful countries of its own time. At first, the empire consisted of only one city (which was the capital) but its people were great warriors so they started to invade their neighbors and increase the area of their lands.
Each time they defeated a neighbor, they razed all their cities except the biggest one, which is added to their kingdom. They built a road from that city to one of their own cities, so that they could travel between all cities of their lands. If the added city was big enough it would become the capital of Mayo.

The vulnerability of a city in Mayo was defined as the number of roads that one had to pass to travel from that city to the capital of Mayo. We have a list of all kingdoms that people of Mayo invaded and the description of the roads that they have built.

We want to know the maximum vulnerability of the cities, after each city was added to Mayo. To simplify the output of the program, report the sum of $v_1 + v_2 + ... + v_n$, where v_i is the maximum vulnerability of the cities after adding the i^{th} city.

Input (Standard Input)

There are several test cases in the input. The first line of each test case contains n ($1 \le n \le 100,000$) which is the number of cities in Mayo. For each test case, the cities are numbered from 1 to n ordered by the time they had been added to Mayo. So, the city 1 is the first city of Mayo (and its initial capital). Each of next n -1 lines of the test case contains two integers j and c. This means that when the city i+1 is added to Mayo, a road is built between the city i+1 and the city j. The number c is non-zero if the new city (city i+1) had become the new capital of Mayo at the time of its join to Mayo, and is zero otherwise. The last line of the input contains a single 0.

Output (Standard Output)

For each test case, output a single line containing the sum of maximum vulnerabilities as explained in the problem description.

Sample Input and Output

Standard Input	Standard Output

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