

Please do not open the problem packet until I tell you to do so.
This packet contains 11 problems lettered A-K.
Feel free to write in this packet. It is yours to keep.

The Navi-Computer is Down!



An Imperial Cruiser and its cohort of Tie Fighters are hot on the tail of the Millennium Falcon. The Falcon has taken minimal damage to the aft and port thrusters, but the Navi-Computer has been damaged preventing a jump to hyperspace and escape.

Luke: *Han, why not just jump to hyperspace? We can't hold up to that Imperial Cruiser much longer!*

Han: *Travelin' through hyperspace ain't like dustin' crops, boy! Without precise calculations we could fly right through a star or bounce too close to a supernova, and that'd end your trip real quick, wouldn't it?*

Leia: *He's right Luke, we need proper calculations entered or...*

C3PO: *We're doomed!*

R2D2: *whistle, chirp, beep, beeeeeeeeeeeeeeeeeeeeeeep!*

Chewbacca: *Groooooooooooooooooaaaaaaaaannnnnn!*

Luke: *I have a very bad feeling about this.*

Leia: *There must be something we can do! We can't go out like this. The Rebel Alliance is depending on us!*

C3PO: *Your Highness, R2D2 says he has a star map of the galaxy with coordinates in 3-space for each location!*

Han: Great, but since the Navi-Computer is down, we need not only the 3-space coordinates, but the distance between our current location and our destination. I'm not up on the tech math to do that and I don't see anyone else here that's a likely candidate.

Leia: *I beg your pardon, but it doesn't take a math tech to do those kinds of calculations...*

Han: *Oh really, your worship? I suppose you're going to tell us they taught you that in 'Princess School'.*

Leia: *Grow up. Anyone that's been to school through the 10th galactic grade knows that the distance between two points, or star systems, in 3-space is the square root of the sum of the difference between the x-coordinates squared, the difference between the y-coordinates squared, and the difference between the z-coordinates squared. The result is in galactic units, of course.*

Luke: *Hey, I remember that! It looks like this* (he draws out the formula):

$$d = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

Han: *Great, so everyone's a genius but me! I'll congratulate you all on your brilliance later...*

Leia (interrupting): *They also teach manners and hygiene in school, but those are usually before grade 4. Looks like you didn't pay much attention, Han.*

Han: ...if you can get me the name and coordinates of the current star system, followed by the name and coordinates of the destination star system, followed by the distance in galactic units between the two, so I can enter it all in the Hyper-Computer and make the jump to hyperspace.

Luke: No problem Han, R2 will provide the coordinates and Leia will do the math, let's get out of here!

Han: Leia, I hope your math is accurate to two digits of precision to the right of the decimal point. The Hyper-Computer will need it.

Your task is to aid Leia in calculating the distance between two star systems with accuracy of two digits to the right of the decimal. Help save the Millennium Falcon and its crew so they can help the Rebel Alliance defeat the Empire!

Input

The input file will begin with a line containing the integer n ($0 < n < 200$), representing the number of entries to process. n entries follow. An entry will consist of the name of the current star system on a line of its own, followed by the x , y , and z coordinates of that star system, all on the same line, all separated by a single space. Next will be the name of the destination star system (which is always different than the current star system) on a line of its own, followed by the x , y , and z coordinates of that star system, all on the same line, all separated by a single space.

Star system names may contain alpha-numeric characters, whitespace, and punctuation. Star system names will be no larger than 30 characters. The x , y , and z coordinates will be real numbers listed with two digits of precision to the right of the decimal point, with $-100,000.00 < x, y, z < 100,000.00$, thus double precision real numbers should be used to avoid rounding errors. Finally, the distance between star systems will always be at least one galactic unit.

Output

For each entry in the input file, output on one line: the name of the current star system, then the string ' to ', then the name of the destination star system, followed by a colon and a space, followed by the distance between the two systems to two digits of precision to the right of the decimal. See the sample output below for details.

Sample Input

```
2
Alderaan
1000.00 2000.00 3000.00
Dantooine
-1000.00 1000.00 1000.00
Circarpous Major
-500.00 500.00 -500.00
Y'Toub
-500.00 -500.00 500.00
```

Sample Output

```
Alderaan to Dantooine: 3000.00
Circarpous Major to Y'Toub: 1414.21
```

Blenjeel Sand Worms and Color Wriggles

The Blenjeel sand worms slither across the sandy surface of the planet Blenjeel. As the planet's only known inhabitants, they protect their homeland by attacking and devouring from underneath anyone who steps foot on their planet.

Of course, the sand worms need to be strong, flexible, and able to slither as quietly as possible. All teenage sand worms are conscripted to a six-month boot camp, where they endure intense training. The most demanding of the training exercise is the famous "wriggle test", which requires worm cadets to slither from one position to a parallel position several hundreds of feet away. Only the toughest, most determined of worms survive.

The exercise is so famous that the Jedi Academy's CS101 has its students solve a puzzle based on Blenjeel Boot Camp. And that puzzle goes something like this:

Given an $n \times m$ board ($3 \leq n \leq 6$, $5 \leq m \leq 50$) wriggle a Blenjeel sand worm (of length n) from the left column to the right column, ensuring the worm never simultaneously occupies two squares of the same color.

Consider the following 3×5 board, where each number represents a different color, and the worm is represented by the chain of circles:

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 4 |
| 3 | 1 | 3 | 1 | 2 |
| 4 | 1 | 4 | 3 | 1 |

One of two possible moves is to pull the bottom of the worm to the right so it's positioned as follows:

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 4 |
| 3 | 1 | 3 | 1 | 2 |
| 4 | 1 | 4 | 3 | 1 |

Now we can pull from the **other** end of the worm to carry it through three different positions:

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 4 |
| 3 | 1 | 3 | 1 | 2 |
| 4 | 1 | 4 | 3 | 1 |

→

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 4 |
| 3 | 1 | 3 | 1 | 2 |
| 4 | 1 | 4 | 3 | 1 |

→

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 4 |
| 3 | 1 | 3 | 1 | 2 |
| 4 | 1 | 4 | 3 | 1 |

Through a series of additional moves, it's possible to wriggle the worm into the target position where the body of the worm is completely in the rightmost column:

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 4 |
| 3 | 1 | 3 | 1 | 2 |
| 4 | 1 | 4 | 3 | 1 |

It's acceptable for the worm to reach the right column in either orientation.

Your job is to write a program that reads in a series of boards as described above, and for each prints out the **minimum** number of wriggles needed to move the worm from the left column to the right (or -1 if there's no solution).

Input

The data will be n rows of m items. In each row will be digits representing a color (colors are represented by the digits 1 through 7 – not all boards will use all 7 colors). The colors in the leftmost column of each input board are guaranteed to be unique. Each board is separated from the next by a blank line. The end of input will be signaled by a standalone 'end'. There will be a blank line between the last board and 'end'.

Output

For each board read, print the minimal number of wriggles needed to move the worm from the left column to the right (or -1 if there's no solution) followed by a newline.

Sample Input

```
12324
31312
41431

234234
342112
421311

234233
342112
421331

41344411122134441231
22313433414323312312
12231221312124143323

41251355234115
13515533543252
34212412323543
52454355242421

364311121136362
151446122112155
434232633624623
561614315456464
234426516251346

end
```

Sample Output

```
16
17
-1
39
31
58
```



This Can't Go On Forever

“A long time ago in a galaxy far, far away...”, so long ago, in fact, that the Empire did not exist and there were planets without space travel. In a far corner was a world in which the predominant pet, called *tayes*, reproduced by budding. Once a *taye* had separated from its parent, it took one time unit to mature to the point of breeding, which, coincidentally, was the length of time for a new bud to grow and separate from its parent. The *tayes* are extremely long-lived, so that it became important to investigate how many someone might have, assuming that at time zero the person did not have a *taye*, and at time one had acquired an immature one, freshly budded from the mature *taye* belonging to a friend.

This ends up giving the following recurrence: $T(0) = 0$, $T(1) = 1$, $T(n) = T(n-1) + T(n-2)$, for $n > 1$.

Computing at the time involved unsigned binary numbers with 24 bits. That motivated a particularly curious inhabitant, Leon, to investigate the series generated by that recurrence, but constrained by a modulus — and not just the modulo 2^{24} forced by computing hardware. So the question becomes how long a series of number is generated by this recurrence, subject to arithmetic with a particular modulus, before it begins repeating. Here are the first few series:

| Modulus | Front end of the series under that modulus | Length of the Period |
|---------|--|----------------------|
| 2 | 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 | 3 |
| 3 | 0 1 1 2 0 2 2 1 0 1 1 2 0 2 2 1 0 1 1 2 0 2 2 | 8 |
| 4 | 0 1 1 2 3 1 0 1 1 2 3 1 0 1 1 2 3 1 0 1 1 2 3 | 6 |
| 5 | 0 1 1 2 3 0 3 3 1 4 0 4 4 3 2 0 2 2 4 1 0 1 1 | 20 |

The problem is to determine the length of the period for each modulus given in the input file and report it.

Input

The input file contains an indeterminate number of lines, each containing a single integer *mod* guaranteed to be in the range $2 \leq \text{mod} \leq 16777216$ (2^{24}). The final line contains a 0 as end of data and should not be processed.

Output

For each modulus in the input file, print the modulus, one blank, and then the size of the smallest period of these *T* numbers under that modulus.

Sample Input

```
2
3
4
5
6
12345678
16777216
0
```

Sample Output

```
2 3
3 8
4 6
5 20
6 24
12345678 700512
16777216 25165824
```

A long time ago in a galaxy far,
far away....

Rescue Beacon

While attempting to evade some old and not-so-happy creditors of his, Han Solo crashed the Millennium Falcon on the ice-world of Hoth. Now he must get some sort of emergency signal working so that he can get rescued before his Wookiee friend becomes a fur-cicle or decides to eat a Hans-kabob. Can you help him?

Han managed to salvage a really bright laser that can be seen from light-years away, and thought that would make a good signal. The downside is that if he just shines it straight up into the sky, there would only be a minimal chance that someone happens to be in the path of the light to see it. Chewbacca, in the meantime, was playing with a bunch of highly-reflective, multi-faceted crystals he found in a cave nearby. Then, in a moment of inspiration, Han realized he could assemble a rescue beacon by shining the light down on the crystal, whose facets will in turn reflect the light into the sky in a multitude of directions!

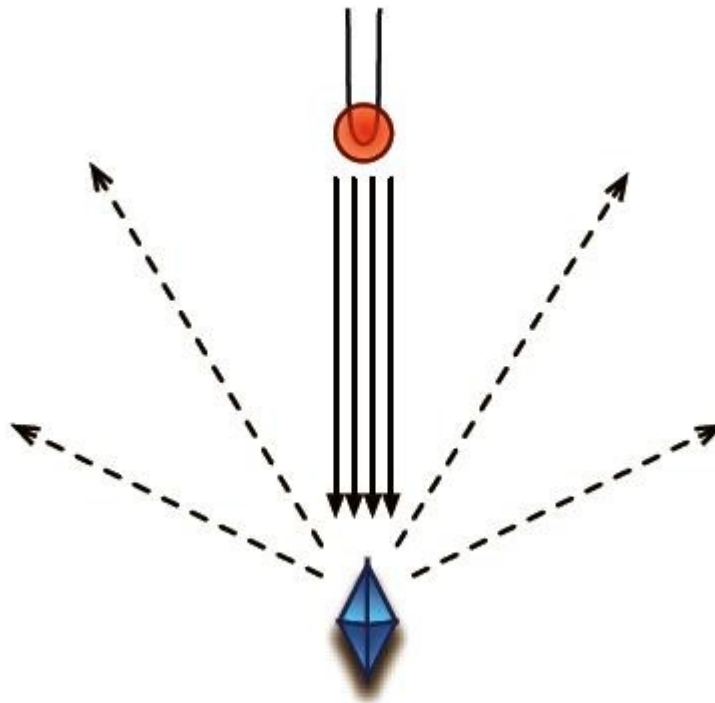


Figure 1: A simplified, two-dimensional illustration of Han's rescue beacon.

The only problem that remains is in deciding which crystal to use as the light reflector. Each of the reflective crystals is a convex polyhedral shape whose surface consists solely of perfectly triangular facets, and can be fitted in the beacon in any orientation. Of course, the best crystal to use is the one that would reflect the laser beam (which consists of parallel light rays from a single direction) back in the greatest number of directions. In other words, you can think of the reflective merit of a crystal as simply the number of facets on it that you can see from any one viewing direction, as those are the facets that can be hit simultaneously by the laser beam. Given a description of the geometry of each of the crystals, can you compute the greatest number of directions the crystal can reflect the laser?

Input

The input will consist of geometric descriptions of the crystals in Chewbacca's collection. The description of each crystal begins with an integer n ($4 \leq n \leq 2000$), the number of facets on the crystal, on a single line followed by n lines describing the facets. Each facet is described by 9 integers, $x_1 y_1 z_1 x_2 y_2 z_2 x_3 y_3 z_3$, where the points (x_1, y_1, z_1) , (x_2, y_2, z_2) , and (x_3, y_3, z_3) in three dimensions form the vertices of the triangular facet in counter-clockwise order (on a right-handed coordinate frame) as viewed from the exterior. All coordinates are in the range $-2000 \leq x_i, y_i, z_i \leq 2000$, no single facet has a surface area greater than 200,000, and no two facets face the same direction. You can expect that when all facets of each crystal are assembled, they form a closed convex polyhedron. Lastly, no crystal will have a structure with any degeneracy that would cause ambiguity as to how many facets the laser can hit. In other words, it should not matter whether you consider facets parallel to the laser beam as capable of reflecting the beam or not -- the test data have been constructed such that either interpretation would yield the same answer. In other words, it should not matter whether you consider facets parallel to the laser beam as capable of reflecting the light or not. Input is terminated by a line containing the number 0 (do not process this as a test case).

Output

For each crystal, output on a single line containing an integer m : the greatest number of directions that the crystal can simultaneously reflect the laser beam.

Sample Input

```
4
0 1 1 -2 0 0 2 0 0
0 1 1 2 0 0 0 3 0
0 1 1 0 3 0 -2 0 0
-2 0 0 0 3 0 2 0 0
6
0 1 1 -2 0 0 2 0 0
0 1 1 2 0 0 0 3 0
0 1 1 0 3 0 -2 0 0
-2 0 0 0 1 -1 2 0 0
2 0 0 0 1 -1 0 3 0
0 3 0 0 1 -1 -2 0 0
0
```

Sample Output

```
3
4
```


Let the Wookiee Win!



You are playing against Chewbacca in a tough battle of Galactic Tic-Tac-Toe, which is just like simple Tic-Tac-Toe except that the playing board is 5×5 and the goal is to get 4 in a row. In response to a brilliant move, Chewbacca protests loudly and plays his next move, setting you up to win. However, before you can claim victory, Han Solo takes this opportunity to warn you that it's not wise to upset a Wookiee. After all, they're known to pull people's arms out of their sockets when they lose. Wisely, you decide on a new strategy: **Let the Wookiee**

win!

Input

Input consists of a series of 5×5 playing boards, each separated by a blank line. Each square in a row is separated by a single space. You are 'O', Chewie is 'X', and empty squares are denoted by '*'. For each board, there is exactly one empty square that you can play that does not cause you to win and does not block any of Chewie's winning moves on his next turn. The final board is followed immediately by the word "Finished" on the next line.

Output

For each board, print a line consisting of the number of the square that you should play to avoid having your arms pulled out of their sockets. The numbering should conform to the following table.

| | | | | |
|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 |

Sample Input

```
X O * X X
O O X X *
O O * X X
O X O O X
X O O X O
```

```
X X X * O
O * * O X
O O X * X
O * X X O
* X O O O
```

Finished

Sample Output

```
3
17
```

Laser Turret Maintenance

As the Chief Weapons Officer of an Imperial Star Destroyer, one of your duties is the routine maintenance of laser turret panels. Each panel is a square composed of $n \times n$ turret sockets, n of which house a turret. To avoid asymmetric wear and tear on the panel circuits, turrets should be regularly moved between sockets. Turrets are capable of rotating in-place in 45° increments, but may not fire in between such rotations. All turrets are aligned with the panel grid upon insertion.

The most important constraint on turret placement is that turrets should not "conflict", meaning they should not be able to shoot each other. Thus, they must be positioned such that no two turrets on the same panel share the same grid horizontal, vertical, or diagonal line. To streamline the process of turret maintenance and reduce placement errors, the Empire has mandated that all turret socket exchanges be performed as rotations or reflections of existing configurations, as conflict prevention is invariant under such transformations.

Given a valid (non-conflicting) $n \times n$ panel configuration, there are three clockwise rotations: 90° , 180° , and 270° . In addition there are four mirror planes: vertical, horizontal, diagonal (across the cells where $row = col$), and anti-diagonal (across the cells where $row + col = n - 1$). Thus there are eight Empire-approved configurations (some of which may be identical for configurations that are symmetric under rotation by 90° and/or 180°). **Figure 1** shows eight approved configurations for $n=5$, while **Figure 2** shows an example from $n=6$ of a configuration with an 180° symmetry and an example from $n=4$ of a configuration with a 90° symmetry.

While the panel is a two-dimensional matrix, the nature of a valid configuration (that there is only one turret in each row) allows it to be represented by a one-dimensional vector, giving the column positions for the turret in each row. The top row of the panel has index zero, similar to the numbering of screen coordinates.

Write a method that receives one valid panel configuration and computes, in the same one-dimensional form, the other seven configurations that are approved by the Empire under the rotations and reflections listed above. Specifically, your program is to write back the configuration received, followed by the configurations obtained by the 90° , 180° , and 270° rotations in that order followed by the reflections in the order vertical mirror, anti-diagonal mirror, horizontal mirror, and diagonal mirror.

Input

Input consists of an indeterminate number of lines consisting of integers separated by white space. The first integer on the line gives the size of the problem (n); zero indicates the end of processing, otherwise $4 \leq n \leq 20$. Following that n will be n integers giving the column positions.

Output

For each problem, print eight lines giving the approved configurations in the order specified above, in which on each line the numbers are right justified in fields of three characters. These eight lines are followed by a blank line, which is shown in Sample Output as '(blank line)'.

Sample Input

```
4 1 3 0 2
5 0 2 4 1 3
6 1 3 5 0 2 4
0
```

Sample Output

```
1 3 0 2
1 3 0 2
1 3 0 2
1 3 0 2
2 0 3 1
2 0 3 1
2 0 3 1
2 0 3 1
(blank line)

0 2 4 1 3
4 1 3 0 2
1 3 0 2 4
2 4 1 3 0
4 2 0 3 1
2 0 3 1 4
3 1 4 2 0
0 3 1 4 2
(blank line)

1 3 5 0 2 4
2 5 1 4 0 3
1 3 5 0 2 4
2 5 1 4 0 3
4 2 0 5 3 1
3 0 4 1 5 2
4 2 0 5 3 1
3 0 4 1 5 2
(blank line)
```

Figure 1: Set of configurations for $n=5$ obtainable by symmetry operations

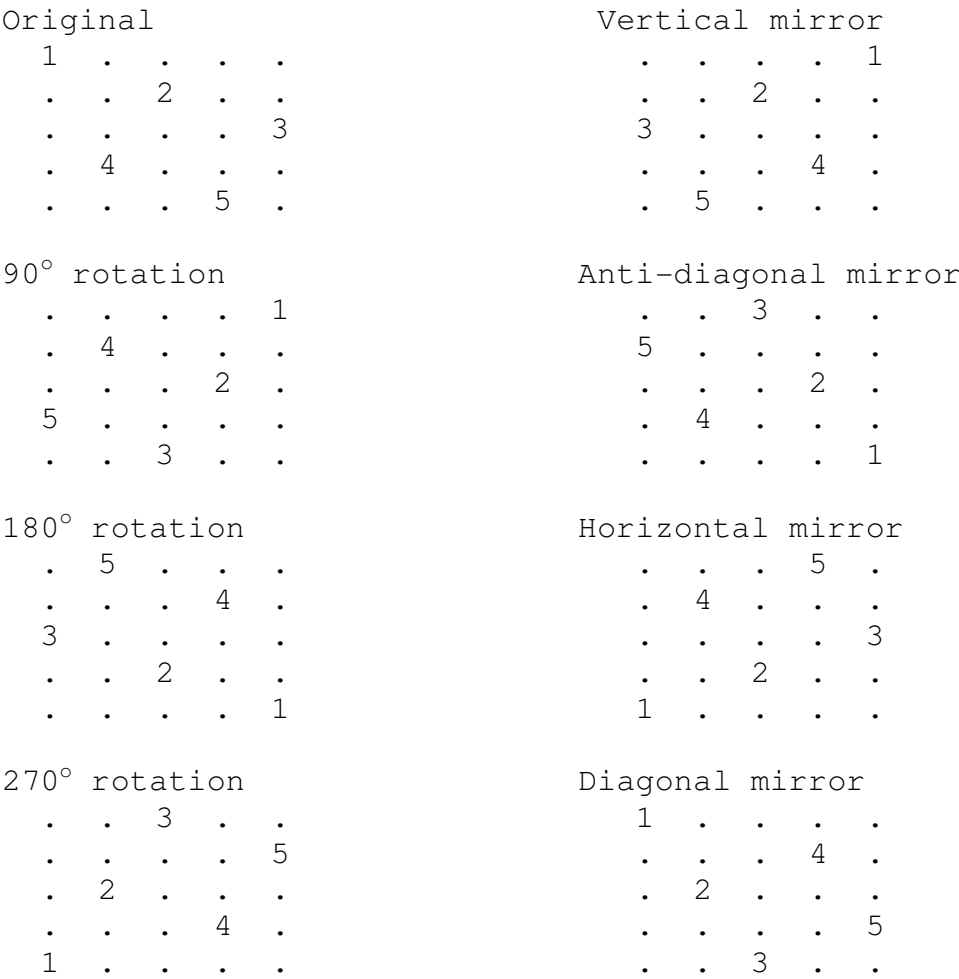
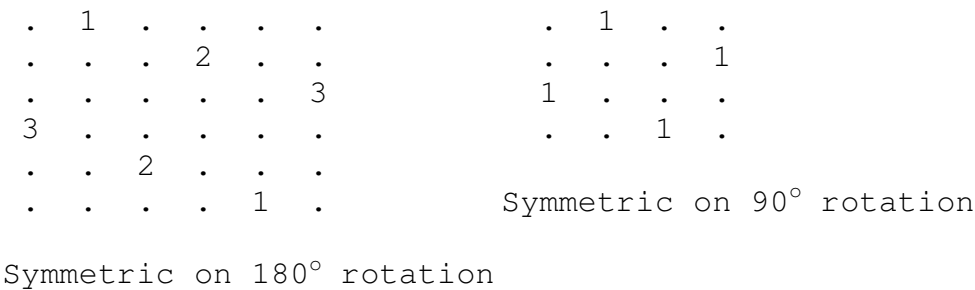


Figure 2: Examples of rotational symmetry in configurations



George Lucas and 1138



It's well known in Star Wars culture that George Lucas was preoccupied with the number 1138. THX 1138 is the title of his first movie, and—according to Wookieepedia: The Star Wars Wiki—George Lucas liked to reuse key words and numbers in later films as a way of unifying all of his works.

If you're up on your Star Wars trivia, then you know that in Episode V a prisoner was transferred from cell block 1138. In Episode II, all clone troopers have the number 1138 prominently displayed on the back of their helmets. And if you type in 1138 while viewing the Episode III DVD, you immediately advance to the scene where Yoda does some break dancing.

Less well known is the unpublished Episode Zero: The Birth of Death Kleene Star. The story begins with C3PO as a tween droid in school on his home planet of Tatooine, where he's been instructed to determine all of the ways one can combine the numbers 1, 1, 3, and 8 using traditional $+$, $-$, $*$, $/$, and $()$ to generate other positive numbers. He's quick to observe that $8 / 3 / 1 - 1$ generates 1, $3 / 8 + 1 + 1$ generates 2, and that $(8 / 3 + 1) * 1$ generates 3. Within seconds, C3PO states that the smallest positive integer that can't be generated is 29. But is C3PO correct?

Given an n digit string ($1 \leq n \leq 7$), compute all of the ways that those n digits can be combined using $+$, $-$, $*$, $/$, and parentheses, and print out the smallest positive integer that can't be generated.

For the string "12345", the numbers 1 through 75 can be generated, but the number 76 can't be. For the string "13555", the numbers 1 through 55 can be generated, but 56 can't be.

Additional Details

- You must use all of the digits, including duplicates.
- You can't combine digits using concatenation, so that $12 + 345$ would **not** be relevant to the generation of all possibilities on behalf of "12345".
- Division is integer division, complete with the truncation you expect from integer division. That means that $4 * (5 / 4)$ is 4, not 5 (although $(4 * 5) / 4$ is, of course, the 5 you'd expect.)
- You must avoid division by zero and disallow it from contributing to the set of generated numbers.
- The test inputs will be such that all intermediate and final results can fit into a 32-bit integer (long).

Input

Expect a series of digit strings, one per line. End of input will be signaled by a 0 on its very own line. Do not process this case.

Output

You should produce one line of output to standard out for every line of input. Each line of output should be the smallest positive integer that **cannot** be generated by some arithmetic expression involving all of the digits.

Sample Input

```
1138
12345
13555
167283
123458
4321678
0
```

Sample Output

```
29
76
56
524
347
1774
```

Spare the Ewoks!

Rejuvenated after the successful siege of Cloud City, Emperor Palpatine has commissioned you to begin construction of a new Imperial base on the forest moon of Endor. Endor, however, is home to a species of cuddly creatures known as the 'Ewoks.'

Having a soft spot in his otherwise callous heart for these Ewoks, Palpatine has ordered that none of the existing Ewok homes must be disturbed. Furthermore, he has asked that you make the base as large as possible in terms of total area, and that this area may be divided among up to three rectangular buildings. Help Palpatine in his quest for intergalactic domination!

For this problem, you are provided a map of Endor in the form of an $m \times n$ rectangular grid, where some of the cells have been marked as Ewok homes, and other cells are empty. Your goal is to place up to three (but possibly fewer) rectangular buildings on this grid in such a way that no two buildings overlap with each other, and no building is placed over an Ewok home.

Input

The input will contain multiple test cases. Each test case begins with a single line containing a pair of integers m and n (where $1 \leq m \leq 250$ and $1 \leq n \leq 250$) representing the number of rows and columns in the grid. The next m lines then specify the map of Endor; specifically, each line will contain n characters, where each character is either '.' (a period) standing for an empty space, or 'e' (lower case E) for an Ewok home. After the final test case will be a single line containing "0 0"; this line should not be processed.

Output

For each input test case, print a single integer indicating the maximum area that can be covered by buildings.

Sample Input

```
1 2
..
5 6
eee...
ee...e
ee...e
e...ee
e...ee
8 12
eee...eee...
eee...eee...
ee...eee...ee
eee...eee...
ee...eee...e
eee...eee...ee
ee...eee...ee
eee...eee...e
0 0
```



Sample Output

```
2
13
22
```


The History of the Sith Rulers



As an archaeologist studying the history of Sith rulers, you regularly need to quickly look up what ruler was in power for a given galactic year. Given a list of Sith rulers and the years they were in power, write a program that will display which ruler (or rulers) ruled for a specified year. Note that no two rulers ever ruled simultaneously. Further note that some years and months had no ruler at all as the empire was in disarray. Finally, some rulers led more than once, but always had a break in ruling before reassuming power (someone else ruled before they reassumed power).

Input

The first entry in the input file will be an integer n ($0 < n \leq 50$) specifying the number of Sith rulers. Following this will be n entries, each representing a Sith ruler. The first line of the entry will contain the name of the ruler, which will be no more than 30 characters in length. The second line of the entry will contain the galactic date the ruler assumed power, followed by a space, followed by the date the ruler left power. Both of these dates will be real numbers greater than 0 and less than 5,000. The second date is guaranteed to be greater than the first date. These real numbers will be listed with one digit of precision to the right of the decimal point. The digit to the right of the decimal point effectively represents a galactic month (starting at 0). For the date the ruler leaves power, the assumption is that the ruler always serves to the end of the month listed. More specifically, another ruler never assumes power the same month the previous ruler leaves power. The minimum time a ruler ever serves is one month (thus a ruler could serve from 1.1 to 1.1, and any ruler that follows must start no earlier than 1.2).

Following the entries for the Sith rulers will be an integer c ($0 < c \leq 50$) specifying the count of year entries that will follow. On the c lines that follow will be a galactic year in integer format. Each year is an integer y , $0 < y < 5000$.

Output

For each galactic year specified in the input file, display the Sith ruler (or rulers) in power during that year in the format shown in the sample output below. More specifically, print “Galactic year”, followed by a space, followed by the year, followed by a colon, followed by a space, followed by the name of the ruler for that year. If there was more than one ruler during that year, subsequent rulers for that year should be listed with a comma and a space preceding them. Furthermore, the rulers should be listed in the order of rule from earliest to latest in that year. If a ruler rules more than one time in a year, that ruler should be listed each time. A newline should immediately follow the last ruler listed. If no rulers are found for a given galactic year, specify “None” following the galactic year designation.

Sample Input

```
8
Tulak Hord
1000.0 1025.9
Ludo Kresh
1026.0 1030.0
Darth Revan
2000.0 2003.0
Darth Malak
2003.1 2009.2
Exar Kun
1500.0 1550.0
Darth Sidious
2500.0 2500.0
Darth Nihilus
2500.1 2500.1
Darth Sidious
2500.2 2500.4
5
1025
2003
1525
1551
2500
```

Sample Output

```
Galactic year 1025: Tulak Hord
Galactic year 2003: Darth Revan, Darth Malak
Galactic year 1525: Exar Kun
Galactic year 1551: None
Galactic year 2500: Darth Sidious, Darth Nihilus, Darth Sidious
```


Laser Shot

You're a droid, and you want to try to shoot a Jedi in the room. It's well known that a Jedi can block a laser with his light saber, so shooting him directly won't work. What you can do, though, is shoot two lasers in such a way that they hit him simultaneously from different angles, in which case he can't block both. Conveniently, the (square) room is mirrored, so you can bounce laser shots off the walls. However, these are not perfectly silvered mirrors, so each laser can only bounce a limited number of times before it dissipates.

As a droid, you have other tasks to complete, so you want to minimize the time spent between firing the two laser shots. You have two laser pistols and droid reflexes, so you can shoot them off at an arbitrarily small interval (including simultaneously).

The room is a square, 1,000,000 feet on each side. Its lower left corner is at $(0, 0)$ and its upper right corner is at $(1,000,000, 1,000,000)$. You are at (x_1, y_1) and the Jedi is at (x_2, y_2) . A laser that bounces around and then passes through (x_1, y_1) will continue along its path (you planned the shot, after all, so you can dodge it), but a laser immediately halts as soon as it hits (x_2, y_2) . A laser fired at a wall follows the normal rules of reflection (angle of incidence = angle of reflection, and reflection introduces no extra delay). A laser fired directly at a corner will rebound exactly opposite the direction it was fired, and will count as having bounced twice. For simplicity, assume the speed of light (and therefore of the laser) is 1 foot per nanosecond.

Input

Each test case consists of a single line of 5 space-separated integers x_1, y_1, x_2, y_2, n , where x_1, y_1, x_2 , and y_2 are as described above, and n is the maximum number of bounces. $1 \leq x_1, y_1, x_2, y_2 \leq 999,999$, and $1 \leq n \leq 100$. You are guaranteed that the droid and Jedi are not in the same location, i.e., $(x_1, y_1) \neq (x_2, y_2)$. The last test case is followed by "0 0 0 0 0", which should not be processed.

Output

For each test case, print on a single line the minimum delay between the firing of the lasers in nanoseconds, accurate to 5 decimal places. Note that the test data have been constructed to ensure that no answers are within $1e-6$ of a rounding boundary.

Sample Input

```
100000 1 100000 999999 1
100000 100000 800000 800000 1
0 0 0 0 0
```

Sample Output

```
19801.94156
0.00000
```



Problem K
Power Grid

Input: k.in

Output: console

After the untimely demise of its previous tenant, you have found yourself promoted to the position of the new “chief architect” of the Death Star by none other than Lord Vader himself. You have been tasked with designing the wiring system for distributing power throughout each sector of the Death Star. Knowing that your life depends on your success, you have decided to be extremely meticulous and exhaustively enumerate all valid wiring configurations so that you may choose the best one.

You are given a map of the Death Star, in the form of an $m \times n$ grid, whose cells correspond to sectors. Exactly one of the sectors in the map is a power station; the remaining sectors are all either living quarters or storage units. You have the option of placing a connection between any two non-storage sectors that are either vertically or horizontally adjacent. A valid wiring configuration is one such that:

- Every living quarter sector is either directly or indirectly connected to the power station sector, and
- As few connections are used as possible (i.e., removing any connection would result in disconnecting some living quarter from the power station).

For a given map, how many valid wiring configurations exist?

Input

The input will contain multiple test cases. Each test case begins with a single line containing a pair of integers m and n (where $1 \leq m \leq 8$ and $1 \leq n \leq 8$). The next m lines each contain n characters, representing the map of the Death Star. The map will contain exactly one sector marked as a power station (P); the remainder of the sections will be either living quarters (.) or storage units (#). You are guaranteed that there exists at least one valid wiring configuration. The end-of-input is denoted by a single line containing “0 0” and should not be processed.

Output

For each input test case, print the number of valid wiring configurations mod 1,000,000,000.

Sample Input

```
2 2
..
.P
4 5
#####
#.P.#
#...#
#####
3 4
....
P#..
...#
6 6
#####
##..##
#....#
#..P.#
##..##
#####
0 0
```

Sample Output

```
4
15
31
768
```

