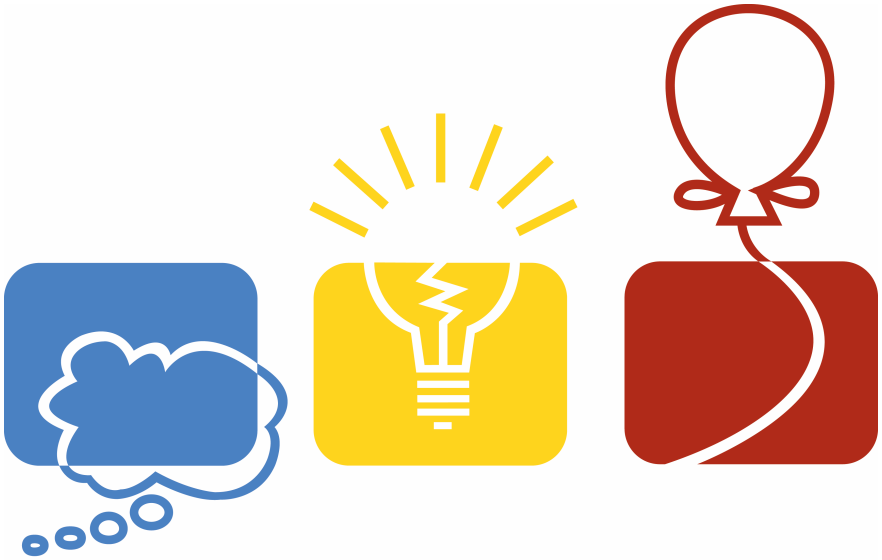


acm International Collegiate Programming Contest

2011 Pacific Northwest Region Programming Contest

Problems

Problem A — limit 5 seconds

Good or Bad?

Description

Bikini Bottom has become inundated with tourists with super powers. Sponge Bob and Patrick are trying to figure out if a given character is good or bad, so they'll know whether to ask them to go jelly-fishing, or whether they should send Sandy, Mermaid Man, and Barnacle Boy after them.

SPONGE BOB: Wow, all these characters with super powers and we don't know whether they are good guys or bad guys.

PATRICK: Well, it's easy to tell. You just have to count up the number of g's and b's in their name. If they have more g's, they are good, if they have more b's, they are bad. Think about it, the greatest hero of them all, Algorithm Crunching Man is good since he has two g's and no b's.

SPONGE BOB: Oh, I get it. So Green Lantern is good and Boba Fett is bad!

PATRICK: Exactly! Uh, who's Boba Fett?

SPONGE BOB: Never mind. What about Superman?

PATRICK: Well he has the same number of g's as b's so he must be neutral.

SPONGE BOB: I see, no b's and no g's is the same number. Very clever Patrick! Well what about Batman? I thought he was good.

PATRICK: You clearly never saw *The Dark Knight*...

SPONGE BOB: Well what about Green Goblin? He's a baddy for sure and scary!

PATRICK: The Green Goblin is completely misunderstood. He's tormented by his past. Inside he's good and that's what counts. So the method works!

SPONGE BOB: Patrick, you are clearly on to something. But wait, are you saying that Plankton is neutral after all the terrible things he's tried to do to get the secret Crabby Patty formula?

PATRICK: Have any of his schemes ever worked?

SPONGE BOB: Hmmm, I guess not. Ultimately he's harmless and probably just needs a friend. So sure, neutral works for him.

PATRICK: Alright then, let's start taking names and figure this out.

SPONGE BOB: But Patrick, if we start counting all day, Squidward will probably get annoyed and play his clarinet and make us lose count.

PATRICK: Well, let's hire a human to do it for us on the computer. We'll pay them with Crabby Patties!

SPONGE BOB: Great idea Patrick. We're best friends forever!

Help Sponge Bob and Patrick figure out who is good and who is bad.

Input

The first line will contain an integer n ($n > 0$), specifying the number of names to process. Following this will be n names, one per line. Each name will have at least 1 character and no more than 25. Names will be composed of letters (upper or lower case) and spaces only. Spaces will only be used to separate multiple word names (e.g., there is a space between Green and Goblin).

Output

For each name read, display the name followed by a single space, followed by " is ", and then followed by either "GOOD", "A BADDY", or "NEUTRAL" based on the relation of b's to g's. Each result should be ended with a newline.

Sample Input	Sample Output
8 Algorithm Crunching Man Green Lantern Boba Fett Superman Batman Green Goblin Barney Spider Pig	Algorithm Crunching Man is GOOD Green Lantern is GOOD Boba Fett is A BADDY Superman is NEUTRAL Batman is A BADDY Green Goblin is GOOD Barney is A BADDY Spider Pig is GOOD

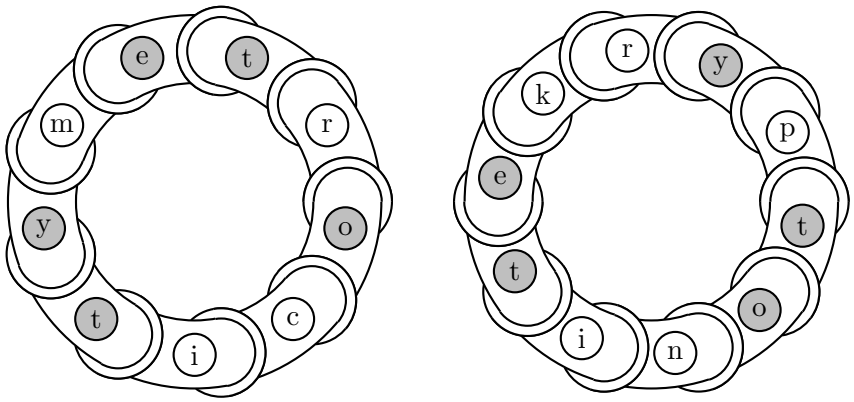
Problem B — limit 30 seconds

Bracelets

Description

Finally, Megamind has devised the perfect plan to take down his arch-nemesis, Metro Man! Megamind has designed a pair of circular power bracelets to be worn on his left and right wrists. On each bracelet, he has inscribed a sequence of magical glyphs (i.e., symbols); each activated glyph augments Megamind's strength by the might of one grizzly bear!

However, there's a catch: the bracelets only work when the subsequences of glyphs activated on each bracelet are identical. For example, given a pair of bracelets whose glyphs are represented by the strings "metrocity" and "kryptonite", then the optimal activation of glyphs would give Megamind the power of 10 grizzly bears:



(a) Left bracelet (b) Right bracelet

Figure 1: Megamind's power bracelets

On the first bracelet, the letters "etoty" are activated in clockwise order; the same letters are activated in counterclockwise order on the second bracelet. Generally, the ordering of the letters is important, but the orientation of the activated subsequence on each bracelet (i.e., clockwise or counterclockwise) may or may not be the same—and don't forget that the bracelets are circular!

Help Megamind defeat Metro Man by determining the optimal subsequences of glyphs needed to activate his bracelets.

Input

The input file will contain at most 100 test cases (including at most 5 "large" test cases). Each test case is given by a single line containing a space-separated pair of strings s and t, corresponding to the sequences of glyphs on Megamind's left and right power bracelets, respectively. Each string will consist of only lowercase letters ('a'-'z'). The length of each input string will be between 1 and 100 characters, inclusive, except for the large test cases where the length of each input string will be between 1 and 1500 characters, inclusive.

Output

For each input test case, print the maximum power (in units of grizzly bears) that Megamind will be able to achieve by activating glyphs on his bracelets.

Sample Input	Sample Output
metrocity kryptonite megamind agemdnim metroman manmetro megamindandmetroman metromanandmegamind	10 16 16 32

Problem C — limit 10 seconds

A Classic Myth: Flatland Superhero

Description

Flatland needs a superhero! Recently swarms of killer ants have been invading Flatland, and nobody in Flatland can figure out how to stop these dastardly denizens. Fortunately, you (as a higher dimensional being) have the opportunity to become a superhero in the eyes of the Flatland citizens! Your job is to "freeze" swarms of ants using parallelograms. You will do this by writing a program that finds a minimal area enclosing parallelogram for each swarm of ants. Once a minimal area parallelogram is placed around the ant swarm, they are effectively frozen in place and can no longer inflict terror on planar inhabitants.

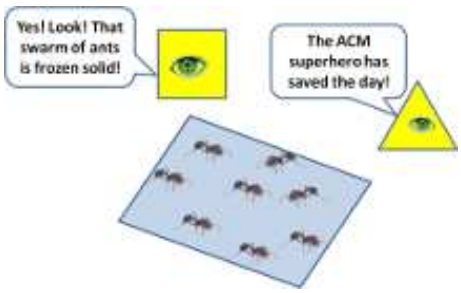


Figure 2: A Frozen Swarm of Ants in Flatland

Input

The input will consist of the following:

- A line containing a single integer, s ($1 \leq s \leq 20$), which denotes the number of killer ant swarms.
- Each swarm will start with a single line containing an integer, n ($4 \leq n \leq 1000$), which indicates the number of killer ants in the swarm.
- The next n lines contain the current location of each killer ant in the swarm.
- Each killer ant is represented by a single line containing two numbers: x ($-1000 \leq x \leq 1000$) and y ($-1000 \leq y \leq 1000$) separated by a space.
- Only one killer ant will occupy each (x, y) location in a particular swarm. Each swarm should be dealt with independently of other swarms.
- All data inputs are in fixed point decimal format with four digits after the decimal (e.g., dddd.dddd).
- There may be multiple parallelograms with the same minimum area.

Output

For each swarm, your algorithm should output a line that contains "Swarm i Parallelogram Area: ", where i ($1 \leq i \leq s$) is the swarm number, followed by the minimum area (rounded to 4 decimal digits and using fixed point format) of an enclosing parallelogram for that swarm. All computations should be done using 64 bit IEEE floating point numbers, and the final answers displayed in fixed point decimal notation and rounded to four decimal digits of accuracy as shown in the sample input and output.

Sample Input

```
2
6
0.0000 0.0000
-0.5000 -0.5000
-1.0000 0.0000
-0.7000 -7.0000
-1.0000 -1.0000
0.0000 -1.0000
5
2.0000 2.0000
0.0000 0.0000
0.5000 2.0000
1.0000 1.0000
1.5000 0.0000
```

Sample Output

```
Swarm 1 Parallelogram Area: 7.0000
Swarm 2 Parallelogram Area: 3.0000
```

Problem D — limit 30 seconds

Collateral Cleanup

Description

Remember the big fight where the Hulk and the Abomination threw each other through the buildings of Manhattan? Or the time when the Green Goblin smashed poor Spider-Man through a good half-dozen brick walls? Wow, they must have shattered those walls into a *million* pieces!!!

It's great that we have superheroes to bring the villains to justice, but have you ever wondered who gets to repair all the collateral damage when they're done? Well actually, as president of the Action Cleanup Management (ACM) corporation, your job is to do exactly that! After a big fight, you must take all the broken pieces of the walls and put them back together just as they were before the fierce battle began.

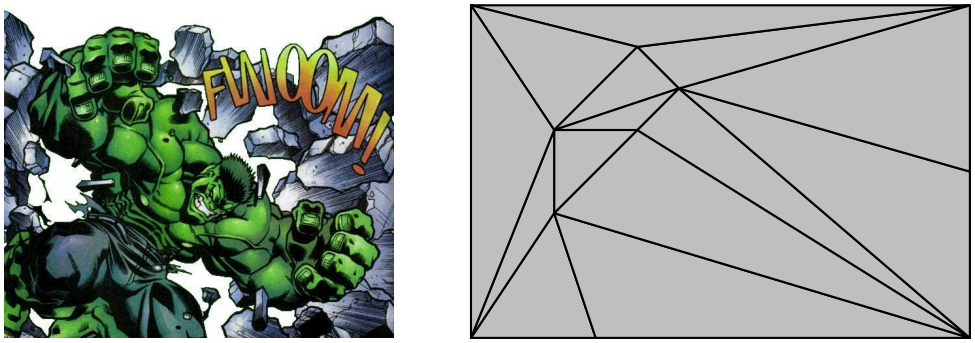
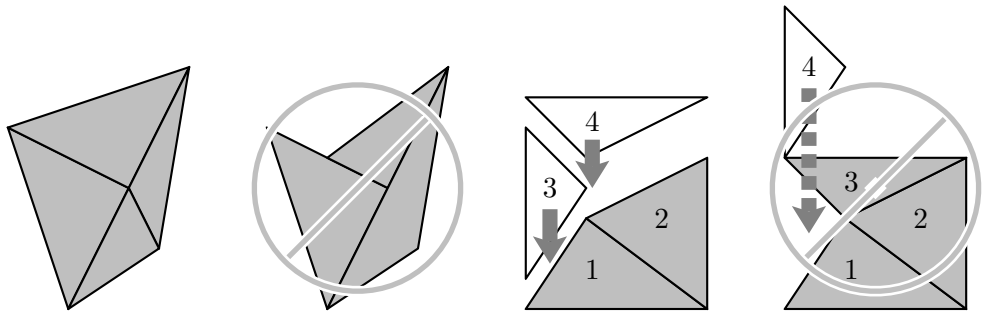


Figure 3: A wall broken into many pieces.

A wall is a perfectly rectangular region that shatters into perfectly triangular pieces when a villain is sent through it (see Figure 3). Through sophisticated visual analysis, you have ascertained where in the original structure every little piece came from. In essence, you have a blueprint that looks a lot like the picture above. Furthermore, you observe that wherever two broken pieces meet, they meet along the full length of the break (edge) that separates them, as shown in Figure 4a.



(a) Breaks that can occur

(b) Breaks that never occur

(c) Ordering that works

(d) Last piece is blocked

Figure 4: The good, the bad, and the ugly.

You have an assembly robot that can help you reconstruct a wall in place. However, the robot can only lower each piece, one at a time, straight down from the top. The robot cannot move a piece from side to side or rotate it in any way to get it where it needs to go. Thus, you must be careful regarding the order in which you tell the robot to reassemble the broken pieces, lest you inadvertently block a piece from being lowered into its proper place (see Figure 4d). Can you determine an ordering of the pieces for each wall that will allow you to fully reassemble it?

Input

An integer on the first line of the input file indicates the number of walls you must reassemble. The first line for each wall has an integer, n, indicating the number of triangular pieces the wall was broken into ($2 \leq n \leq 1{,}000{,}000$). Then, n lines of input follow, each describing a piece with six integers, $x_1\ y_1\ x_2\ y_2\ x_3\ y_3$, that correspond to the Cartesian (x, y) coordinates of the three corners of a triangle on the original wall from which the piece came. The first line describes piece 1, the next line piece 2, and so forth. The three points will always be given in a counterclockwise winding order and form a triangle of non-zero area. All coordinates lie between 0 and 10^9 inclusive, with the positive y direction being the "up" direction. The n pieces given will cover a rectangular region exactly, with no gaps or overlaps.

Output

For each wall, output on a single line the numbers of the pieces, separated by spaces, in an order that will allow your robot to reassemble the wall. If more than one correct solution exists, any ordering that will work is acceptable.

Sample Input

```
2
4
3 4 7 1 7 6
7 6 1 6 3 4
1 6 1 1 3 4
7 1 3 4 1 1
14
0 0 3 0 2 3
2 3 3 0 12 0
2 3 12 0 4 5
4 5 12 0 5 6
5 6 12 0 12 4
5 6 12 4 12 8
5 6 12 8 4 7
4 7 12 8 0 8
4 7 0 8 2 5
4 7 2 5 5 6
5 6 2 5 4 5
4 5 2 5 2 3
2 3 2 5 0 0
0 0 2 5 0 8
```

Sample Output

```
4 1 3 2
1 2 13 14 3 4 12 11 5 6 10 9 7 8
```

Problem E — limit 10 seconds

LatticeLand

Description

LeaperLad wakes in LatticeLand on a disk suspended above a lake of lava. Leaper spies his HeloPak on one of the disks; with it he knows he can escape this nefarious trap.

The disks are quite far apart, however; without some momentum, he can only jump to an immediately adjacent disk. Once he has acquired the speed to make that jump, he can accelerate on every disk he touches.

He notices the disks are laid out in a rectangular grid, with a disk on each grid point. He calculates that on each disk he can accelerate or decelerate his speed by one unit in the horizontal or vertical direction (but not both on the same disk). Alternatively, he can just maintain his speed when stepping on a disk. Thus, in a straight line, from a standing start, he can jump one unit, then two units, then three, then two, then one.

Some pairs of disks are joined by walls of fire that he knows he must not touch. He can get arbitrarily close to one of these walls, but he must not touch one. Nor can he fall off the edge of the grid.

How quickly can LeaperLad reach his HeloPak and stop on that disk?

Input

Input will have one problem per input line. The input line will contain a sequence of integers, each separated by a single space.

The first two integers will be w and h, the width and height of the grid. Each of these values will be between 1 and 64, inclusive. Following that will be two integers representing the coordinates of the disk that LeaperLad wakes on. After that will be two integers representing the coordinates of the disk that the HeloPak is on. The next integer will be f, the number of fire walls. There will be six or fewer fire walls. After that will be f sets of 4 integers, representing the two coordinates of the end points of the walls.

For all coordinates, the first number will be between 0 and $w - 1$, inclusive, and the second number will be between 0 and $h - 1$, inclusive. All fire walls will be at least one unit long. The HeloPak and LeaperLad will never start on the same disk, nor will either start on a disk that is on a firewall. There will always be a way for LeaperLad to reach his HeloPak.

There will be no more than 50 problems.

Output

For each input line, print a single integer indicating the minimal number of moves needed for LeaperLad to reach his HeloPack. Pay close attention to the first couple of examples; they clarify how moves are counted.

Examples

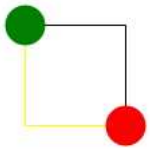
Example 1

```
2 1 0 0 1 0 0
```

This requires two moves. In the first move, LeaperLad accelerates one unit in the positive x direction, and hops onto the destination disk. In the second move, he decelerates to the required speed of zero (note that although LeaperLad's position does not change during the second move, it nonetheless counts towards the total).

Example 2

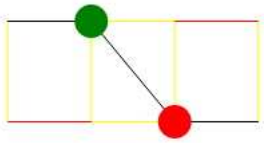
```
2 2 0 0 1 1 0
```



This requires four moves. LeaperLad first moves to the right, as in the prior example, but he must decelerate in the x direction first, then accelerate in the y direction to jump down, then decelerate again to become motionless.

Example 3

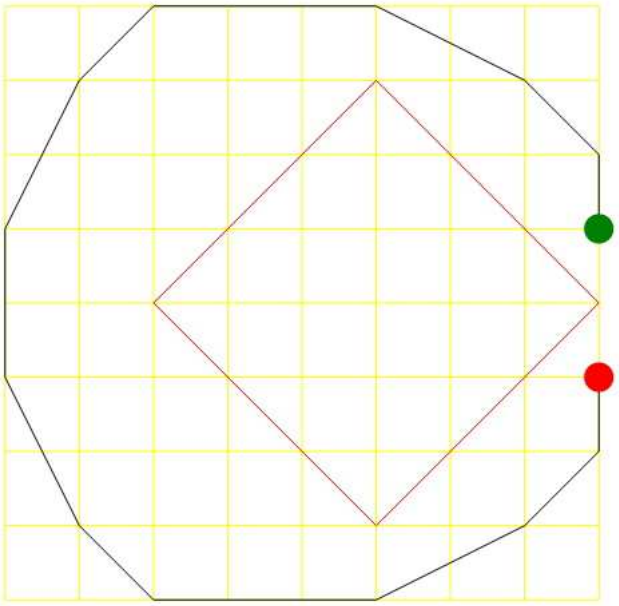
```
4 2 1 0 2 1 2 0 1 1 1 2 0 3 0
```



This requires eight moves. LeaperLad cannot jump diagonally from a standing stop; he needs to back up to get some momentum first. So first he must move to position $(0, 0)$, then decelerate to turn around, then accelerate in the x direction while jumping to his original location. His momentum allows him to accelerate in the y direction to make a diagnoal move. Once landing at his destination, he has one unit of momentum in both x and y directions, so he must decelerate, first in the y direction (which takes him to position $(3, 1)$, overshooting his destination). Then he decelerates to turn around, jumps to his destination once more, and then decelerates to be motionless, for a total of eight moves.

Example 4

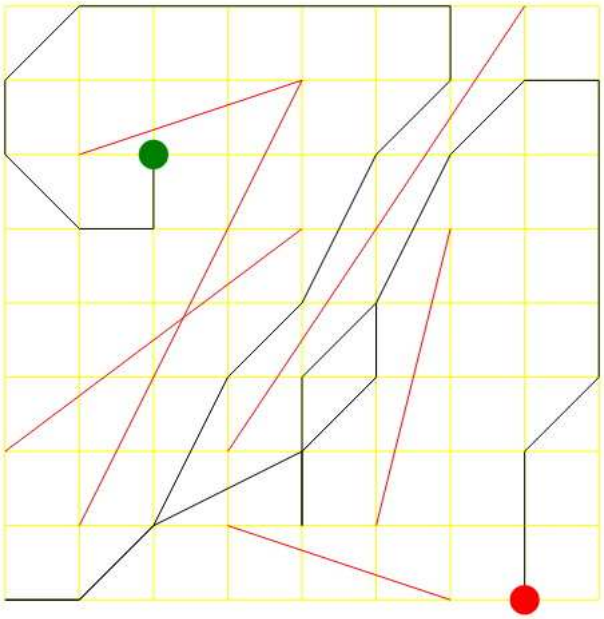
```
9 9 8 3 8 5 4 8 4 5 1 5 1 2 4 2 4 5 7 5 7 8 4
```



This requires 16 moves.

Example 5

```
9 9 2 2 7 8 6 0 6 4 3 6 8 3 7 1 7 4 1 3 6 7 0 4 1 1 2 5 7 6 3
```



This requires 43 moves.

Sample Input	Sample Output
2 1 0 0 1 0 0 2 2 0 0 1 1 0 4 2 1 0 2 1 2 0 1 1 1 2 0 3 0 9 9 8 3 8 5 4 8 4 5 1 5 1 2 4 2 4 5 7 5 7 8 4 9 9 2 2 7 8 6 0 6 4 3 6 8 3 7 1 7 4 1 3 6 7 0 4 1 1 2 5 7 6 3	2 4 8 16 43

Problem F — limit 5 seconds

Lightning Lessons

Description

Zeus wrung his hands nervously. "I've come to you because I agreed to duel Thor in the upcoming Godfest. You're good in a fight, Raiden; you've got to help me!"

Raiden, smiling thinly beneath the rim of his hat, replied, "What help could I provide a god as mighty as yourself? Your thunderbolts are the stuff of legends!" Zeus looked down and stammered, "I've... I've been lucky. I don't know how the thunderbolts actually work. Sometimes I turn my foe into a charred heap, but other times... weird stuff happens. If Apollo hadn't convinced the bards to keep my secret, I'd be a laughingstock."

Raiden raised his eyebrows and asked, "Weird stuff?" Zeus looked up and took a deep breath. "Sometimes it just fizzles out. Other times it rolls up and turns into a... a bunny." Raiden burst out laughing. "A bunny! That's some chi you've got there." As Zeus began to redden, Raiden held up his hand and said, "Don't worry, I'll help you out."

Raiden went on to explain. "A thunderbolt is a sequence of chi pivots, or 'zigs and zags' as the mortals call them. Each pivot has an integer amplitude—"

"Yes, I know that much.", Zeus interrupted. "But lightning is lively and unpredictable. The amplitudes go all random once the bolt hits!"

"Not all that flickers is flame. If you watch the bolt closely, you'll see it goes through 'cycles', and gets shorter by one pivot each cycle. When the bolt cycles, each successive pivot's amplitude is decreased by the amplitude of its predecessor from the end of the previous cycle, and the first pivot vanishes. If a bolt ever reaches a state of all zero amplitudes, it converges and zaps its target with power proportional to the number of preceding cycles. Your 'weird stuff' happens only when a bolt cycles down to a single non-zero amplitude. A positive amplitude just fizzles out into waste heat, but negative amplitudes produce odd low-entropy states. It's the latter you've seen hopping away in the midst of battle."

Help Zeus avoid embarrassment by writing a program that predicts how powerful a given bolt will be if it converges, or what will happen to it if it diverges.

Input

The first line of input contains a single positive integer N which denotes how many lightning bolts follow. Each bolt is specified by a line beginning with an integer M ($0 < M \leq 20$), followed by M space-delimited integers denoting the initial amplitudes of each successive pivot. No initial amplitude will have an absolute value larger than 1000.

Output

For each bolt that converges, output the letter "z" repeated P times, where P is the number of cycles encountered before the bolt converges, followed by the string "ap!" (the all-zero cycle does not count toward P).

For each bolt that fails to converge, output "*fizzle*" if the final amplitude was positive, "*bunny*" if it was negative.

Sample Input	Sample Output
4 2 1 1 5 1 3 6 10 15 5 1 2 4 8 16 2 1 0	zap! zzzap! *fizzle* *bunny*

Problem G — limit 30 seconds

Locksmith

Description

Mooks A, C, and M have been charged with creating locks for the Riddler's lair. They need to design locks comprised of flat interlocking pieces that can be unlocked only by separating these pieces while sliding them around on the surface of the doors. Unfortunately, while the mooks have come up with quite a few designs, they are not very good at determining which designs can actually be unlocked.

The pieces of the lock are axis-aligned polygons (that is, the sides are all either horizontal or vertical) in the plane. The lock can be manipulated by sliding any one polygon at a time in any direction, so long as doing so does not cause the polygon to overlap with any other polygon. Rotating a polygon is not allowed. A polygon is considered separable from the rest of the lock if there exists a sequence of moves (possibly involving several polygons) leading to a configuration such that it is possible to draw a straight line between the polygon and the remainder of the lock. Given a proposed lock configuration, your job is to determine the number of separable polygons.

Input

The input file will contain multiple test cases. Each test case begins with a line with a single integer N denoting the number of pieces in the lock. This line is followed by N lines of space-separated integers each with the following format:

$c \quad x_1 \quad y_1 \quad \cdots \quad x_c \quad y_c$

Here, c denotes the number of vertices in the piece, and x_i and y_i give the locations of the vertices in clockwise order. The sides of each piece are all either horizontal or vertical, and they alternate between the two. The pieces are guaranteed to have no overlapping area. Each test case has either 2 or 3 pieces, and for each test case, these pieces have a total of at most 30 vertices. All coordinates given are integers between 0 and 1000 inclusive. Input will be terminated by a case with 0 pieces, which should not be processed.

Output

For each input test case, print the number of separable pieces in the lock.

Sample Input	Sample Output
2 12 0 0 0 6 9 6 9 0 6 0 6 1 8 1 8 5 1 5 1 1 3 1 3 0 4 2 2 2 4 7 4 7 2 2 12 0 0 0 6 9 6 9 0 6 0 6 1 8 1 8 5 1 5 1 1 3 1 3 0 4 4 2 4 4 7 4 7 2 0	0 2

Problem H — limit 5 seconds

Speed Racer

Description

Speed Racer must go go go rescue Trixie at the top of Mount Domo! He must get there as quickly as possible, but his Mach 5 only holds a specific amount of fuel, and there is no way to refuel on the way. Luckily, he knows precisely how much fuel is consumed at a particular speed, taking into account air resistance, tire friction, and engine performance. For a given speed v in kilometers per hour, the amount of fuel consumed in liters per hour is

$$av^4 + bv^3 + cv^2 + dv$$

Assuming Speed travels at a constant speed, his tank holds t liters of fuel, and the top of Mount Domo is m kilometers away, how fast must he drive?

Input

The input will be one problem per line. Each line will contain six nonnegative floating point values representing a, b, c, d, m, and t, respectively. No input value will exceed 1000. The values of c, d, m, and t will be positive. There will always be a solution.

The output should be formatted as a decimal number with exactly two digits after the decimal point, and no leading zeros. The output value should be such that the Speed Racer will not run out of fuel (so truncate, rather than round, the final result). You are guaranteed that no final result will be within 10^{-6} of an integer multiple of 0.01.

Output

The required output will be a single floating point value representing the maximum speed in kilometers per hour that Speed Racer can travel to reach the top of Mount Domo without running out of gas.

Sample Input	Sample Output
0.000001 0.0001 0.029 0.2 12 100 2.8e-8 7.6e-6 0.0013 0.47 11.65 20.81 1.559e-7 1.8195e-5 0.0022233 0.31292 58.902 85.585	134.41 257.45 142.65

Problem I — limit 10 seconds

The Status is Not Quo

Description

Dr. Horrible desperately wants to get into the Evil League of Evil but is having a difficult time proving his competence as the mastermind that he is. Bad Horse rules over the league with an iron hoof and is evaluating his application with extreme skepticism. Meanwhile, arch-nemesis Captain Hammer, hero of the people and corporate tool, is making life exceedingly complicated for our poor villain. But, everything is about to change. Dr. Horrible is all set to pull off a major heist; the wondeflonium needed to complete work on his freeze ray is being transported by courier van—candy from a baby. Sadly, it's not as easy as Dr. Horrible suspected. The device he created to control the van became a jumble of wires that needs to be untangled. He'd have Moist, his roommate, do it, but it's probably a bad idea to have Moist anywhere near circuitry (for obvious reasons). You better do it and do it fast, or else it's curtains for you—lacy, gently wafting curtains.

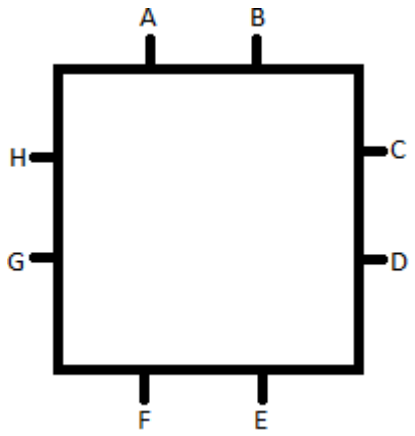


Figure 5: Connection point labels

You can help by figuring out a given wire's ending position based on its starting position for a variety of circuit boards. Here circuit boards are rectangular grids of squares, and each square has 8 connection points, two on each side. Squares also have any number of wires (between 0 and 4 inclusive) connecting one connection point of the square to another. A connection point for a square can only be used by one wire or not be used at all; there is no branching. By naming the square's connection points alphabetically from A to H (always capitalized) starting with the left connection on the top edge and traveling clockwise, you can describe each square as the collection of wires traveling from one named point to another. For example, if a square has a wire traveling from the left connection on the top edge to the bottom connection on the right edge, a wire from the left connection of the bottom edge to the right connection of the bottom edge, and a wire from the right connection of the top edge to the top connection of the left edge then you could describe the square as AD BH EF. For consistency, each wire pair is described alphabetically (BH instead of HB), and all the wire pairs for each square are listed in alphabetical order when describing a square.

Squares are aligned next to each other on all sides to make up the circuit board. For any given square, connection points A and B connect with F and E respectively with the square above it, and vice versa for the square below it. Connection points C and D connect with H and G respectively with the square to its right, and vice versa for the square to its left. If a square has a wire to any given connection point, its corresponding connection point in the adjacent square is guaranteed to continue the path of that wire from its own connection to another connection. There are no broken paths; all paths begin and end at the edge of the circuit board.

Input

Input consists of multiple puzzle sets. Each puzzle set is broken into two parts, a board description and a set of starting points. The board description begins with a single line containing two integers, h and w, both between 1 and 20 inclusive, separated by a space. These are the height and width, respectively, of the circuit board in squares. After this are n ($1 \leq n \leq h \cdot w$) lines containing square descriptions, which occur in no particular order. Each of these line describes one square and begins with a numeric designation for the square. The squares are numbered sequentially left to right, top to bottom, starting at 1; for example, the top right square is numbered w. After the number is the description of the wiring for the square as defined above. The number and all the wire descriptions are separated by single spaces. Not all squares may have a description, and a square will be described at most once per circuit board. Squares without lines have no wires connected to them.

The board description is separated from the set of starting points by a line containing only the number zero ("1"). The starting points are given on the next line, each consisting of a number and a letter together. The starting points are separated from each other by single spaces. The number of the starting point is the square and the letter is the connection point of the square to use. Only connection points on the outside of the circuit board will be given. Only connection points used by a wire will be given. Following this line is a single empty line before the start of the next puzzle set.

The end of the file is marked by two zeros separated by a space in place of the standard first line of a puzzle set.

Output

For each puzzle set, on one line output "Board" followed by a space followed by the number of the board (the number of the first puzzle starts at 1 and increments by 1 for every following puzzle) followed by a colon (":") character.

For each starting point in the set, output the ending point of that wire in the format of "{startpoint} is connected to {endpoint}" on one line. For example, if the given starting point was 1A and the end point was 9H, the output for that starting point would be "1A is connected to 9H". Capitalization matters. All wires are bidirectional, so for the same puzzle, if the starting point was 9H, the output would be "9H is connected to 1A". The letter portion of the start point and end point must be capitalized. There should be no other marks or punctuation.

The output for each puzzle set should be separated by a blank line.

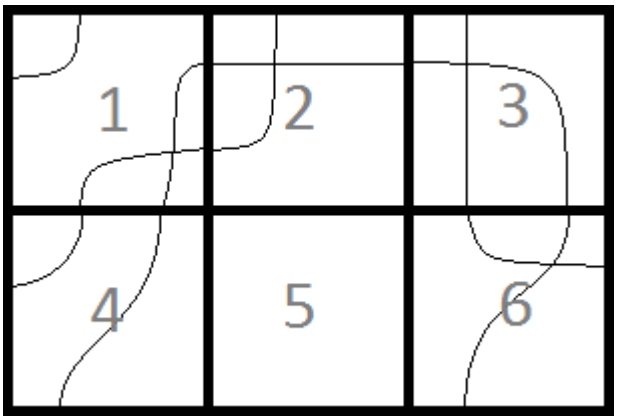


Figure 6: First sample case

Sample Input

```
2 3
4 AG BF
3 AF EH
6 AC BF
1 AH CE DF
2 AG CH
0
1A 2A 3A 4F

4 4
4 DH FG
15 BE FH
3 AD CE
13 BD
5 AD BF
6 BD FG
14 CG
16 AB
1 AE CH FG
7 BG CF DE
11 AG BC EH
9 AG DE
8 AH DG
10 AC DG
2 EH
12 DE FH
0
15F 1G 8D 3A

0 0
```

Sample Output

```
Board 1:
1A is connected to 1H
2A is connected to 4G
3A is connected to 6C
4F is connected to 6F

Board 2:
15F is connected to 3A
1G is connected to 15E
8D is connected to 12D
3A is connected to 15F
```

Problem J — limit 10 seconds

Stupendous Man

Description

Things were grim at Calvin's house. With his parents gone for the night, the nefarious BabySitter Girl had swooped in and taken over with her oppressive regime of evil and terror. But unfortunately for BabySitter Girl, she was unaware that the house was also home to the greatest superhero that the world has ever seen. Up in his room, the mild-mannered Calvin leapt into his closet and emerged as... STUPENDOUS MAN! Champion of Liberty! Foe of Tyranny! With his muscles of magnitude and heroic resolve, Stupendous Man bravely ran downstairs and fought the evil Rosalyn before escaping out the door and into the night. With BabySitter Girl chasing after him, Stupendous Man had only one option: he must evade BabySitter Girl before sneaking back into the house. Should he get caught, Calvin would get in trouble when his parents got home.

Stupendous Man runs in a swirling, twisting, winding loop across the lawn as fast as he can to try to lose BabySitter Girl. His tremendous speed (KAPWINGGG!) means that BabySitter Girl can only see him where his trajectory crosses over itself. To try to catch him, she writes down and assigns a number to each self-crossing in the order which Stupendous Man passes through them (see Figure 7). Sometimes, she makes a mistake, and that's when Calv... I mean Stupendous Man... escapes her grasp! If the numbers Rosalyn wrote down in her observations cannot possibly form a single, closed loop, then she is totally confused, has no idea where Calvin is, and must give up on trying to catch him.

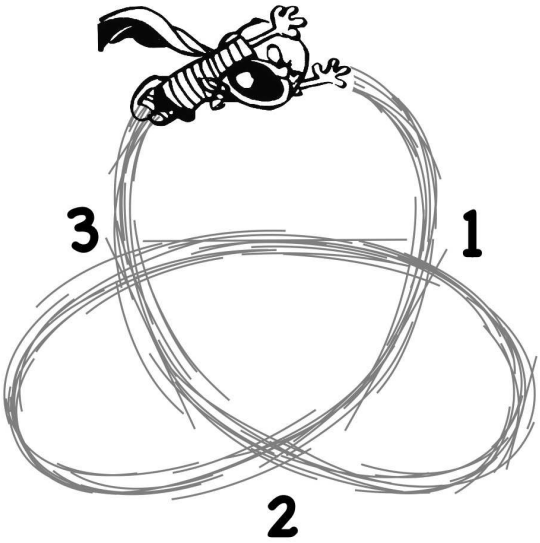


Figure 7: Stupendous Man's trajectory and the three crossings where BabySitter Girl can see him.

Input

The input will consist of multiple test cases. Each test case starts with a line that contains a single positive integer N less than or equal to 10. The next line will contain a sequence of $2N$ integers that Rosalyn has written down based on her observation of Stupendous Man's trajectory. Numbers are separated by a single space, and it is guaranteed that each number from 1 to N appears exactly twice in the sequence. Note that numbers need not appear first in ascending order! The end of input is marked with a line that contains a single zero.

Output

For each test case, print a single line with the word "escaped" if Stupendous Man successfully confuses and evades BabySitter Girl, or "caught" if she manages to untangle his path and find him. Calvin escapes if, and only if, Rosalyn's labeling cannot form a single, closed loop.

Sample Input	**Sample Output**
2 2 1 2 1 3 1 2 3 1 2 3 5 1 2 3 4 5 1 4 5 2 3 0	escaped caught escaped

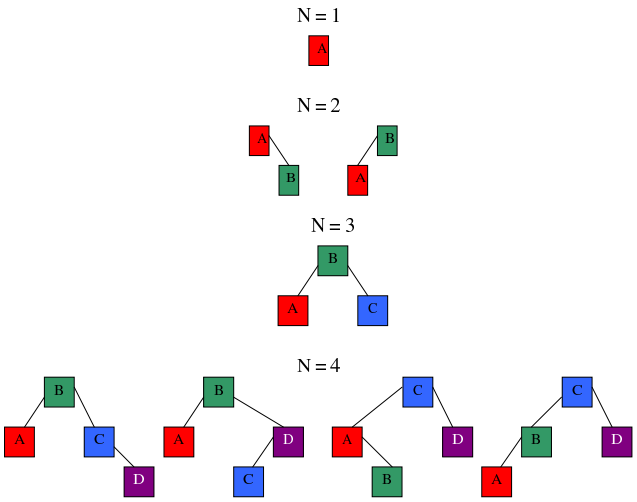
Problem K — limit 5 seconds

Tree Count

Description

Our superhero Carry Adder has uncovered the secret method that the evil Head Crash uses to generate the entrance code to his fortress in Oakland. The code is always the number of distinct binary search trees with some number of nodes, that have a specific property. To keep this code short, he only uses the least significant nine digits.

The property is that, for each node, the height of the right subtree of that node is within one of the height of the left subtree of that node. Here, the height of a subtree is the length of the longest path from the root of that subtree to any leaf of that subtree. A subtree with a single node has a height of 0, and by convention, a subtree containing no nodes is considered to have a height of -1.



Input

Input will be formatted as follows. Each test case will occur on its own line. Each line will contain only a single integer, N, the number of nodes. The value of N will be between 1 and 1427, inclusive.

Output

Your output is one line per test case, containing only the nine-digit code (note that you must print leading zeros).

Sample Input	Sample Output
1 3 6 21	000000001 000000001 000000004 000036900