Problem A: Ambiguous permutations

Some programming contest problems are really tricky: not only do they require a different output format from what you might have expected, but also the sample output does not show the difference. For an example, let us look at permutations.

A **permutation** of the integers *1* to *n* is an ordering of these integers. So the natural way to represent a permutation is to list the integers in this order. With $n = 5$, a permutation might look like 2, 3, 4, 5, 1.

However, there is another possibility of representing a permutation: You create a list of numbers where the *i*-th number is the position of the integer *i* in the permutation. Let us call this second possibility an **inverse permutation**. The inverse permutation for the sequence above is 5, 1, 2, 3, 4.

An **ambiguous permutation** is a permutation which cannot be distinguished from its inverse permutation.

The permutation 1, 4, 3, 2 for example is ambiguous, because its inverse permutation is the same. To get rid of such annoying sample test cases, you have to write a program which detects if a given permutation is ambiguous or not.

Input Specification

The input contains several test cases.

The first line of each test case contains an integer n ($1 \leq n \leq 100000$). Then a permutation of the integers *1* to *n* follows in the next line. There is exactly one space character between consecutive integers. You can assume that every integer between *1* and *n* appears exactly once in the permutation.

The last test case is followed by a zero.

Output Specification

For each test case output whether the permutation is ambiguous or not. Adhere to the format shown in the sample output.

Sample Input

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

Sample Output

```
ambiguous
not ambiguous
ambiguous
```

Problem B: Bullshit Bingo

Bullshit Bingo is a game to make lectures, seminars or meetings less boring. Every player has a card with 5 rows and 5 columns. Each of the 25 cells contains a word (the cell in the middle has always the word "BINGO" written in it). Whenever a player hears a word which is written on his card, he can mark it. The cell in the middle is already marked when the game starts. If a player has marked all the words in a row, a column or a diagonal, he stands up and shouts "BULLSHIT". After this, the game starts over again.

Sitting in a lecture, you observe that some students in the audience are playing Bullshit Bingo. You wonder what the average number of different words is until "BULLSHIT" is exclaimed. For the purpose of this problem, a word consists of letters of the English alphabet ('a' to 'z' or 'A' to 'Z'). Words are separated by characters other than letters (for example spaces, digits or punctuation). Do the comparison of words case-insensitively, i.e., "Bingo" is the same word as "bingo". When counting the number of different words, ignore the word BULLSHIT (indicating the end of the game), and consider only the words of the current game, i.e., if a word has already occurred in a previous game, you may still count it in the current game. If the last game is unfinished, ignore the words of that game.

Input Specification

The input file consists of the text of the lecture, with "BULLSHIT" occurring occasionally. The first game starts with the first word in the input. Each occurrence of "BULLSHIT" indicates the end of one game. You may assume, that

- the word "BULLSHIT" occurs only in uppercase letters
- every word has at most 25 characters, and each line has at most 100 characters
- there are at most 500 different words before a game ends
- the players follow the rules, so there is no need to check if a game is valid or not

Output Specification

The output consists of one number: the average number of different words needed to win a game. Write the number as a reduced fraction in the format shown below. Reduced fraction means that there should be no integer greater than 1 which divides both the numerator and denominator. For example if there were 10 games, and the number of different words in each game summed up to 55, print "11 / 2".

Sample Input

```
Programming languages can be classified BULLSHIT into following types:
- imperative and BULLSHIT procedural languages
- functional languages
- logical BULLSHIT programming languages
- object-oriented BULLSHIT languages
```

Sample Output

```
9 / 2
```

Problem C: 106 miles to Chicago

In the movie "Blues Brothers", the orphanage where Elwood and Jack were raised may be sold to the Board of Education if they do not pay 5000 dollars in taxes at the Cook Country Assessor's Office in Chicago. After playing a gig in the Palace Hotel ballroom to earn these 5000 dollars, they have to find a way to Chicago. However, this is not so easy as it sounds, since they are chased by the Police, a country band and a group of Nazis. Moreover, it is 106 miles to Chicago, it is dark and they are wearing sunglasses. As they are on a mission from God, you should help them find the safest way to Chicago. In this problem, the safest way is considered to be the route which maximises the probability that they are not caught.

Input Specification

The input file contains several test cases.
Each test case starts with two integers n and m ($2 \le n \le 100$, $1 \le m \le n*(n-1)/2$). n is the number of intersections, m is the number of streets to be considered.
The next m lines contain the description of the streets. Each street is described by a line containing 3 integers a, b and p ($1 \le a, b \le n$, $a \ne b$, $1 \le p \le 100$): a and b are the two end points of the street and p is the probability in percent that the Blues Brothers will manage to use this street without being caught. Each street can be used in both directions. You may assume that there is at most one street between two end points.
The last test case is followed by a zero.

Output Specification

For each test case, calculate the probability of the safest path from intersection *1* (the Palace Hotel) to intersection n (the Honorable Richard J. Daley Plaza in Chicago). You can assume that there is at least one path between intersection *1* and n.
Print the probability as a percentage with exactly 6 digits after the decimal point. The percentage value is considered correct if it differs by at most 10^{-6} from the judge output. Adhere to the format shown below and print one line for each test case.

Sample Input

```
5 7
5 2 100
3 5 80
2 3 70
2 1 50
3 4 90
4 1 85
3 1 70
0
```

Sample Output

```
61.200000 percent
```

Problem D: Decorate the wall

After building his huge villa, Mr. Rich cannot help but notice that the interior walls look rather blank. To change that, he starts to hang paintings from his wonderful collection. But soon he realizes that it becomes quite difficult to find a place on the wall where a painting can be placed without overlapping other paintings. Now he needs a program which would tell him, given the already placed paintings, where to place the next painting without moving any other paintings (or indicating that this is impossible). Paintings have a rectangular shape and are to be placed parallel to the side of the wall. If you do not mind a nice reward from Mr. Rich, go on and solve the problem.

Input Specification

The first line of the input file contains a number representing the number of test cases to follow. Each test case starts with a line containing three numbers n, w and h. n is the number of paintings already hanging on the wall, w is the width of the wall and h is the height of the wall. The next n lines contain 4 integers x_1, y_1, x_2, y_2 each ($0 \le x_1 < x_2 \le w$, $0 \le y_1 < y_2 \le h$); the x-coordinates give the distance to the left end of the wall, the y-coordinates give the distance to the bottom of the wall. (x_1, y_1) is the position of the lower left corner of a painting, (x_2, y_2) is the position of the upper right corner. The last line of each test case contains the dimensions of the next painting to be placed, first its width w', then its height h' ($1 \le w' \le w$, $1 \le h' \le h$). You are not allowed to rotate the painting.
You can assume that $0 \le n \le 200$ and $1 \le w, h \le 1000000$. Moreover, all paintings already hanging do not overlap.

Output Specification

Produce one line of output for each test case. Write "Fail!" if there is no place left on the wall where the painting could be placed without overlapping other paintings. Otherwise, write the coordinates where the lower left corner of the painting should be placed. In case there is more than one solution, select the solution with a minimum y-coordinate, and break ties using the minimum x-coordinate.

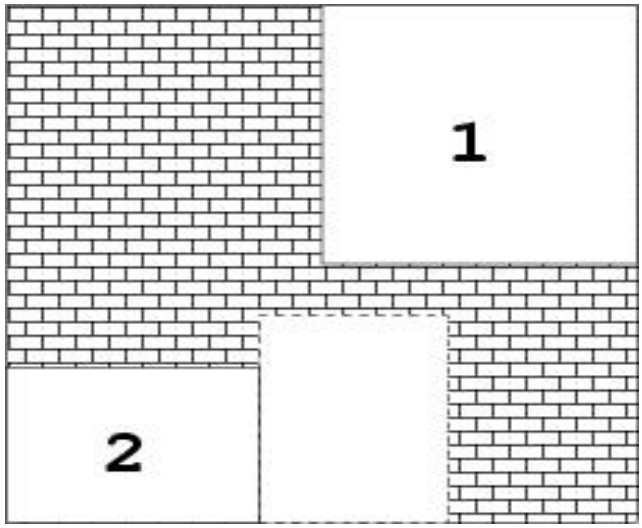
Sample Input

```
2
1 10 9
5 4 10 9
9 5
2 10 10
5 5 10 10
0 0 4 3
3 4
```

Sample Output

```
Fail!
4 0
```

The following image illustrates the second sample test case:



Problem E: Any fool can do it

Surely you know someone who thinks he is very clever. You decide to let him down with the following problem:

- "Can you tell me what the syntax for a set is?", you ask him.
- "Sure!", he replies, "a set encloses a possibly empty list of elements within two curly braces. Each element is either another set or a letter of the given alphabet. Elements in a list are separated by a comma."
- "So if I give you a word, can you tell me if it is a syntactically correct representation of a set?"
- "Of course, any fool can do it!" is his answer.

Now you got him! You present him with the following grammar, defining formally the syntax for a set (which was described informally by him):

```
Set          ::= "{" Elementlist "}"
Elementlist  ::= <empty> | List
List         ::= Element | Element "," List
Element      ::= Atom | Set
Atom         ::= "{" | "}" | ","
```

`<empty>` stands for the empty word, i.e., the list in a set can be empty.
Soon he realizes that this task is much harder than he has thought, since the alphabet consists of the characters which are also used for the syntax of the set. So he claims that it is not possible to decide efficiently if a word consisting of `"{"`, `"}"` and `","` is a syntactically correct representation of a set or not.
To disprove him, you need to write an efficient program that will decide this problem.

Input Specification

The first line of the input file contains a number representing the number of lines to follow.
Each line consists of a word, for which your program has to decide if it is a syntactically correct representation of a set. You may assume that each word consists of between *1* and *200* characters from the set { `"{"`, `"}"`, `","` }.

Output Specification

Output for each test case whether the word is a set or not. Adhere to the format shown in the sample output.

Sample Input

```
4
{}
{{}}
{{}},{,}}
{,,}
```

Sample Output

```
Word #1: Set
Word #2: Set
Word #3: Set
Word #4: No Set
```

Problem F: Cubic Rube

Working in a Rubik's Cube factory has always been your dream job, but you're a clumsy rube. Now you're trying to invent a new variation, a 5x5x5 Rubik's Cube! You just walked into the research lab with a working prototype and accidentally dropped it, breaking it into 2 pieces. You have one piece in your left hand and are trying to find the other. Unfortunately, the floor is littered with parts from previous attempts, making it difficult to find the missing piece. Luckily, you know how to program...

Given two solids composed of unit cubes, determine if they can be fit together to form a solid 5x5x5 cube.

Input:

The first line contains a single integer *n* indicating the number of data sets. The following lines contain the data sets.

Each data set will show side-by-side top-down views of each of the solids as a 5x5 array of numbers 0-9. Each number represents the height of the solid at that point (the number of unit cubes stacked at that point). If the height is 0, there is no solid present at that point. This is similar to providing a topographical map of each solid, and it is valid to assume that there are no gaps "under" any part of a solid that cannot be seen.

Note that each solid will be connected (i.e., a single piece) and may require translations (not just rotations) to join with a compatible solid.

Output:

For each data set, print, "Yes" if a solid 5x5x5 cube can be formed by fitting together the two halves, and print, "No" if a cube cannot be formed. Each word should appear on its own line.

Sample Input:

```
2
55551 11111
55551 11111
55551 11111
55551 11110
55552 00000
22222 33333
22222 33333
22222 33233
22222 33333
22222 33333
```

Sample Output:

```
Yes
No
```