2005/2006 ACM International Collegiate Programming Contest
University of Ulm Local Contest

Problem A: Ambiguous permutations

Source file: ambiguous.(c|cc|hs|java|pas)
Input file: ambiguous.in

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
```

2005/2006 ACM International Collegiate Programming Contest
University of Ulm Local Contest

Problem B: Bullshit Bingo

Source file: bingo.(c|cc|hs|java|pas)
Input file: bingo.in

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
```

In the sample input, there are 4 completed games. The number of different words is 5, 5, 4 and 4, respectively.

2005/2006 ACM International Collegiate Programming Contest
University of Ulm Local Contest

Problem C: 106 miles to Chicago

Source file: chicago.(c|cc|hs|java|pas)
Input file: chicago.in

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 ($2 \leq n \leq 100$, $1 \leq m \leq n*(n-1)/2$). 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 ($1 \leq a, b \leq n$, $a \neq b$, $1 \leq p \leq 100$): 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
```

The safest path for the sample input is 1 -> 4 -> 3 -> 5

2005/2006 ACM International Collegiate Programming Contest
University of Ulm Local Contest

Problem D: Decorate the wall

Source file: decorate.(c|cc|hs|java|pas)
Input file: decorate.in

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 ($0 \leq x_1 < x_2 \leq w$, $0 \leq y_1 < y_2 \leq h$); 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' ($1 \leq w' \leq w$, $1 \leq h' \leq h$). You are not allowed to rotate the painting.
You can assume that $0 \leq n \leq 200$ and $1 \leq w, h \leq 1000000$. 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.

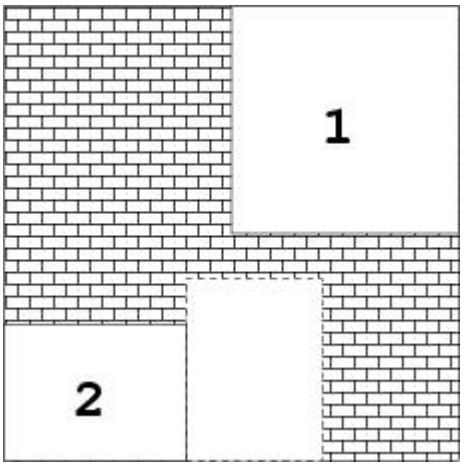
Sample Input

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

The following image illustrates the second sample test case:

Sample Output

```
Fail!
4 0
```



2005/2006 ACM International Collegiate Programming Contest
University of Ulm Local Contest

Problem E: European railroad tracks

Source file: european.(c|cc|hs|java|pas)
Input file: european.in

As you may already know, different countries in Europe use different railroad systems. Not only do they use different voltages for their trains, but also the distance between the two rails (gauge) differs. The following table shows some railway gauges used:

Broad gauge (Spain):	**1674 mm**
Broad gauge (Portugal):	**1665 mm**
Broad gauge (Ireland):	**1600 mm**
Broad gauge (Finland):	**1524 mm**
Broad gauge (former USSR):	**1520 mm**
Standard gauge:	**1435 mm**
Narrow gauge (meter gauge):	**1000 mm**

A museum has trains from several countries. It needs tracks for every train type in order to show visitors the trains in use. However, since only one train is used at a time, a rail can be used by trains of different types. It follows that for n trains, each requiring a different railway gauge, $n + 1$ rails are sufficient (each train uses the leftmost rail and a rail that has exactly the required distance to it). But sometimes it is possible to save even more rails.

Given the required railway gauges, your task is to construct a railway track that can be used by every train and requires the least number of rails. Note that a train can use any two rails, provided the distance between them is right.

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 an integer n (the number of different railway gauges required). The next line contains n integers between 1000 and 5000, each defining one required railway gauge.
You can assume that $1 \leq n \leq 8$. Moreover, for every test case in the input file, there will be a solution requiring at most 5 rails.

Output Specification

The output for each test case consists of three lines:
The first line is of the form "Scenario #X", where X is the test case number starting with 1. The second line describes the solution your program has found; first your program should print how many rails are needed, followed by a colon, then the positions of each rail in increasing order (the first rail should be at position 0). The third line should be blank. If there are several solutions with the minimum number of rails, any one will do.

Sample Input

```
3
4
1524 1520 1609 1435
3
1000 1520 1600
6
1000 2000 3000 4000 1500 2500
```

Sample Output

```
Scenario #1
4: 0 1520 1609 3044

Scenario #2
4: 0 1000 1520 1600

Scenario #3
5: 0 1500 3000 4000 5000
```

2005/2006 ACM International Collegiate Programming Contest
University of Ulm Local Contest

Problem F: Any fool can do it

Source file: fool.(c|cc|hs|java|pas)
Input file: fool.in

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
```

2005/2006 ACM International Collegiate Programming Contest
University of Ulm Local Contest

Problem G: Game schedule required

Source file: game.(c|cc|hs|java|pas)
Input file: game.in

Sheikh Abdul really loves football. So you better don't ask how much money he has spent to make famous teams join the annual tournament. Of course, having spent so much money, he would like to see certain teams play each other. He worked out a complete list of games he would like to see. Now it is your task to distribute these games into rounds according to following rules:

- In each round, each remaining team plays at most one game
- If there is an even number of remaining teams, every team plays exactly one game
- If there is an odd number of remaining teams, there is exactly one team which plays no game (it advances with a wildcard to the next round)
- The winner of each game advances to the next round, the loser is eliminated from the tournament
- If there is only one team left, this team is declared the winner of the tournament

As can be proved by induction, in such a tournament with n teams, there are exactly $n - 1$ games required until a winner is determined.
Obviously, after round 1, teams may already have been eliminated which should take part in another game. To prevent this, for each game you also have to tell which team should win.

Input Specification

The input file contains several test cases. Each test case starts with an integer n ($2 \leqslant n \leqslant 1000$), the number of teams participating in the tournament. The following n lines contain the names of the teams participating in the tournament. You can assume that each team name consists of up to *25* letters of the English alphabet ('a' to 'z' or 'A' to 'Z').
Then follow $n - 1$ lines, describing the games the sheikh would like to see (in any order). Each line consists of the two names of the teams which take part in that game. You can assume that it is always possible to find a tournament schedule consisting of the given games.
The last test case is followed by a zero.

Output Specification

For each test case, write the game schedule, distributed in rounds.
For each round, first write "Round #X" (where X is the round number) in a line by itself. Then write the games scheduled in this round in the form: "A defeats B", where A is the name of the advancing team and B is the name of the team being eliminated. You may write the games of a round in any order. If a wildcard is needed for the round, write "A advances with wildcard" after the last game of the round, where A is the name of the team which gets the wildcard. After the last round, write the winner in the format shown below. Print a blank line after each test case.

Sample Input

```
3
```

2005/2006 ACM International Collegiate Programming Contest
University of Ulm Local Contest

Problem H: Help the problem setter

Source file: help.(c|cc|hs|java|pas)
Input file: help.in

Preparing a problem for a programming contest takes a lot of time. Not only do you have to write the problem description and write a solution, but you also have to create difficult input files. In this problem, you get the chance to help the problem setter to create some input for a certain problem.
For this purpose, let us select the problem which was not solved during last year's local contest. The problem was about finding the optimal binary search tree, given the probabilities that certain nodes are accessed. Your job will be: given the desired optimal binary search tree, find some access probabilities for which this binary search tree is the unique optimal binary search tree. Don't worry if you have not read last year's problem, all required definitions are provided in the following.
Let us define a **binary search tree** inductively as follows:

- The empty tree which has no node at all is a binary search tree
- Each non-empty binary search tree has a root, which is a node labelled with an integer, and two binary search trees as left and right subtree of the root
- A left subtree contains no node with a label $\geqslant$ than the label of the root
- A right subtree contains no node with a label $\leqslant$ than the label of the root

Given such a binary search tree, the following **search procedure** can be used to locate a node in the tree:
Start with the root node. Compare the label of the current node with the desired label. If it is the same, you have found the right node. Otherwise, if the desired label is smaller, search in the left subtree, otherwise search in the right subtree.
The **access cost** to locate a node is the number of nodes you have to visit until you find the right node.
An **optimal binary search tree** is a binary search tree with the minimum expected access cost.

Input Specification

The input file contains several test cases. Each test case starts with an integer n ($1 \leqslant n \leqslant 50$), the number of nodes in the optimal binary search tree. For simplicity, the labels of the nodes will be integers from *1* to n. The following n lines describe the structure of the tree. The i-th line contains the labels of the roots of the left and right subtree of the node with label i (or -1 for an empty subtree). You can assume that the input always defines a valid binary search tree.
The last test case is followed by a zero.

Output Specification

For each test case, write one line containing the access frequency for each node in increasing order of the labels of the nodes. To avoid problems with floating point precision, the frequencies should be written as integers, meaning the access probability for a node will be the frequency divided by the sum of all frequencies. Make sure that you do not write any integer bigger than $2^{63} - 1$ (the maximum value fitting in the C/C++ data type *long long* or the Java data type *long*). Otherwise, you may produce any solution ensuring that there is exactly one optimal binary search tree: the binary search tree given in the

input.

Sample Input

```
3
-1 -1
1 3
-1 -1
10
-1 2
-1 3
-1 4
-1 5
-1 6
-1 7
-1 8
-1 9
-1 10
-1 -1
0
```

Sample Output

```
1 1 1
512 256 128 64 32 16 8 4 2 1
```

Note that the first test case in the sample input describes a tree looking like

```
  2
 / \
1   3
```

```
A
B
C
A B
B C
5
A
B
C
D
E
A B
C D
A E
C E
0
```

Sample Output

```
Round #1
B defeats A
C advances with wildcard
Round #2
C defeats B
Winner: C

Round #1
A defeats B
C defeats D
E advances with wildcard
Round #2
E defeats A
C advances with wildcard
Round #3
E defeats C
Winner: E
```

Note that there is always more than one possible game schedule; you may print any of them.