MERCER UNIVERSITY

Spring Programming Competition
February 25, 2012

Adopt-a-Hydrant	(prob1)
Calculator Games	(prob2)
Changing	(prob3)
Continued Fractions	(prob4)
Dice Game	(prob5)
Numbers in Circles	(prob6)
Palindromic Sum	(prob7)
Please, Let Me Count the Ways	(prob8)
Prefix Free Set	(prob9)
Raindrops	(prob10)
Treasure Maps	(prob11)
Who Wants to be a Code for America City?	(prob12)

The name in parenthesis following each problem is the name you must use as your program's name. You must add the appropriate extension depending upon your choice of programming language (.c .cpp .cs .java .py .rb).

Adopt-a-Hydrant (prob1)

One of the most successful recent projects of Code for America ("A New Kind of Public Service," see more at codeforamerica.org) is the Adopt-a-Hydrant program which allows individuals to take responsibility for fire hydrants in their neighborhoods. These people will be responsible for notifying authorities if anything is wrong with the hydrant. They also will ensure the hydrant is easily accessible in case it is needed. For example, in some cities, snow fall can hide hydrants completely. (Aren't you glad you don't live there?) For this program, you will create a simple version of the Adopt-a-Hydrant program.

Input

The input for this program will consist of multiple input sets. Each input set will begin with a pair of integers, *hydrants*, $0 \leqq hydrants \leqq 200$ indicating the number of hydrants in the town and *people*, $0 \leqq people \leqq 1000$ the number of people who are volunteering to adopt them. There will then be *people* lines, each beginning with the name of a person, consisting of 1 or more alphanumeric characters (['a'-'z', 'A' -'Z', '0' - '9']), then a list of 1 to 20 integers in the range 1 .. *hydrants*, indicating the hydrants the person is volunteering to adopt in their order of preference. People will be given hydrants in the order of their preference. There will be at most one person assigned to each hydrant and each person will be responsible for at most one hydrant. The first person gets their first choice of hydrant. The second will get their first choice, unless the first person has already selected that one, in which case they will get their second choice of hydrant, and so on. It is possible a person will not be able to be assigned a hydrant from their list.[1]
Once a person has been assigned a hydrant, the assignment will not change.
End of input will be indicated by a line with *hydrants* = 0 and some value for *people*. No matter what the value for people is on this line, there may be no additional data in the input. There should be no output produced by this line.

Output

The output for each input set should start with a line indicating the number of the input set, beginning with 1. There should then be *hydrants* lines, each giving the number of the hydrant, consecutively numbered starting at 1, a colon, and the name of the person who adopted it or nothing if no one adopted it.

[1] Yes, this is a simple algorithm. If you'd like to try something more complex, maybe you should apply for Code For America.

Following the list of hydrants, there should be a line consisting of the string “Unable to adopt” and then a list of the people who were not assigned a hydrant, one name per line in the order they appeared in the input set. If everyone was assigned a hydrant, the title should not be printed.
There should be a blank line after the output for each input set.

Sample Input

```
4 3
Jessica 1 2
Nick 1
Zach    1 4    2
2 1
Bob 2 1
0 4
```

Sample Output

```
Set 1
1: Jessica
2:
3:
4: Zach
Unable to adopt
Nick

Set 2
1:
2: Bob
```

Calculator Games (prob2)

The Problem

You are given a calculating device that has a single display screen that displays non-negative integers less than 1 billion (up to 9 digits) and three buttons. The three buttons perform the following functions on the current value, x, on the display screen:

(1) Add 1
(2) Multiply by 3
(3) Integer Division by 2

You are given a starting positive integer n < 100 and your goal will be to figure out the minimum number of button presses necessary to reach all of the other positive integers less than 100. Your program should simply output the maximum of all of these numbers, namely, the most number of button presses needed to reach any particular value.
For example, if the starting value were 1, in one button press, the numbers 2 (1 + 1), 3 (1 * 3), and 0 (1 / 2) would be reached. In two button presses, the values 3, 6, 1, 4, and 9 would be reached.

Input

The first line will contain a single positive integer, n < 100, specifying the number of input cases.
Each input case will have a single positive integer, k < 100, on a line by itself representing the starting value for that case.

Output

For each input case, on a line by itself, output the value of the input case followed by a colon and a space and then the maximum number of button presses to reach any of the numbers from 1 to 99 for that case.

Sample Input

```
3
1
73
99
```

Sample Output

```
1: 10
73: 11
99: 12
```

Changing (prob3)

Change is hard.

Change is also heavy, when it's in your pocket. When you go shopping, you may want to spend as much change as possible. Suppose you have one quarter, three dimes, two nickels, and 20 pennies in your pocket. You could pay for a 30 cent purchase using a quarter and nickel, but that would leave you with 24 coins. If you pay for it with two nickels and 20 pennies, you'll only have four coins left.

Of course, things may not always go well. If you have one quarter and three dimes, when you pay for a two cent purchase, you'll get more change. You should assume the change you receive back will be given in as few coins as possible. If you use the quarter, you'll have 8 coins, five dimes and three pennies, at the end. If you use a dime, you'll have seven coins, one quarter, two dimes, a nickel, and three pennies at the end. Since you want to minimize the number of coins, you should pay with a dime.

Sometimes overpayment (coins that sum to more than the amount of the purchase) is necessary, but no overpayment that has a proper subset that is greater than or equal to the purchase amount is permitted. For example, if you have a purchase amount of $1, you cannot pay for it with four quarters and 5 pennies (in order to get a nickel back) since the subset of just 4 quarters can be used to pay for the purchase.

Write a program that will help you minimize the number of coins you have after making a purchase.

Input

The input will consist of one or more data sets.

The first line of each data set will be four non-negative integers, representing the number of pennies, nickels, dimes, and quarters available, in that order.

The next lines of the data set will represent the amounts of the purchases to be made in cents, one per line. Each purchase should start with the original group of coins. The end of the list of amounts will be a purchase of 0. This line should not be processed.

The end of data is indicated by a data set that begins with a line containing all zeros. This line should not be processed.

Output

The output for each data set should start with a line indicating the number of the data set, beginning with 1. There should then be one line for each purchase amount, each giving the purchase amount, a colon, and either the number and type of coins that should be used to pay for it to minimize the final number of coins, along with the final number of coins, or a message indicating that the purchase cannot be made because it is too large.
There should be a blank line after the output for each data set.

Sample Input

```
20 2 3 1
30
45
196
0
0 0 3 1
2
54
0
0 0 0 0
```

Sample Output

```
Data Set 1:
30: 20 pennies, 2 nickels, 0 dimes, 0 quarters (4 coins left)
45: 20 pennies, 1 nickels, 2 dimes, 0 quarters (3 coins left)
196: not enough change

Data Set 2:
2: 0 pennies, 0 nickels, 1 dimes, 0 quarters (7 coins left)
54: 0 pennies, 0 nickels, 3 dimes, 1 quarters (1 coins left)
```

Continued Fractions (prob4)

Continued fractions are a method of representing real numbers as a sum of integer fractions. In fact, this method of representation is oftentimes preferred because it can sometimes provide a clearer, more concise representation of a number than a decimal representation.
The form of a continued fraction is as follows:

$$r = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots}}}$$

With this representation, any real number r can be represented as a sequence {a0 , a1 , a2 , ...} of integers. All rational numbers have a finite length representation. In fact, they actually have two. One of the representations will be longer than the other.
For instance, 17 can be represented as the following continued fraction:

$$\frac{17}{14} = 1 + \cfrac{1}{4 + \cfrac{1}{1 + \cfrac{1}{2}}}$$

Or for another example

$$\frac{1024}{543} = 1 + \cfrac{1}{1 + \cfrac{1}{7 + \cfrac{1}{1 + \cfrac{1}{3 + \cfrac{1}{7 + \cfrac{1}{2}}}}}}$$

Your task is, given a rational number represented as a fraction, calculate the minimum length continued fraction sequence for that number.

Input

The first line of input will contain an integer N , telling how many cases will follow. The next N lines will contain a pair of positive integers representing a fraction in the following format: n/d where n is the numerator and d is the denominator. There will be at most 1000 cases.

Output

For each case presented, you should print a comma delimited list of the continued fraction coefficients for the given fraction.

Sample Input

```
3

17/14

1024/543

31/14
```

Sample Output

```
1, 4, 1, 2

1, 1, 7, 1, 3, 7, 2

2, 4, 1, 2
```

Dice Games (prob5)

The Problem

Recently, Mercer-Bradley has announced a number of board game contests. In each of these games, contestants have to throw dice. After carefully reading the rules of all the different games, you realize that all of the games use in between one and five dice, inclusive, and that all the dice used are always numbered from 1 to n, where n represents the number of sides on that particular die. Of course, the dice are fair, so each side is equally likely to occur.

Mercer-Bradley is offering a great deal of prize money for winning these various games in a tournament. Naturally, you want to maximize your chances of winning by being able to compute the probability of rolling any target sum, given the appropriate dice configuration. Your job, before the tournament, will be to write a computer program that automates the task, so you can claim the Mercer-Bradley prize money for yourself!

Input

The first line will have a positive integer, n, representing the number of input cases. Each case follows, one input case per line. The first value for each input case will be a positive integer, ($d < 6$), representing the number of dice for the problem. The following *d* integers on the line will represent the number of sides on each of the dice. These values will all be chosen from the set {4, 6, 8, 12, 20}. This will be followed by a single integer, T, representing the target value for that case.

Output

For each input case, output the number of the input case, followed by a colon, one space, and probability of rolling the target exactly, rounded to 9 decimal places.

Sample Input

```
2

2 6 6 2

4 6 8 20 4 33
```

Sample Output

```
1: 0.027777778

2: 0.013541667
```

Numbers in Circles[2] (prob6)

The Problem

In figure 1, each circle contains a number. Each pair of circles is joined by a line on which is written the sum of the numbers in the two circles. That is, 3 + 8 = 11, 3 + 7 = 10 and 7 + 8 = 15. In figure 2, the number on each line again shows the sum of the numbers in the circles joined by that line, but the numbers have been left out of the circles. Can you work out what the missing numbers are?

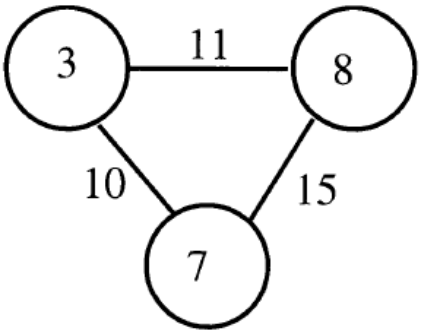


Figure 1

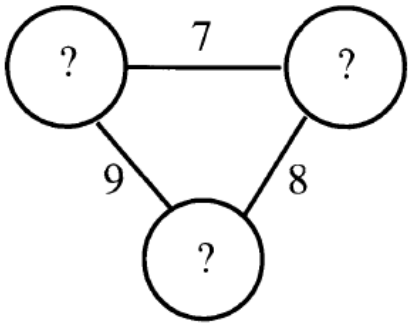


Figure 2

Extending the Problem

Suppose that instead of writing the sums of the numbers on the lines, we were to write their products. For instance figure 3 shows that A × B = 2, B × C = 8, C × D = 12, D × E = 15, and E × A = 5.

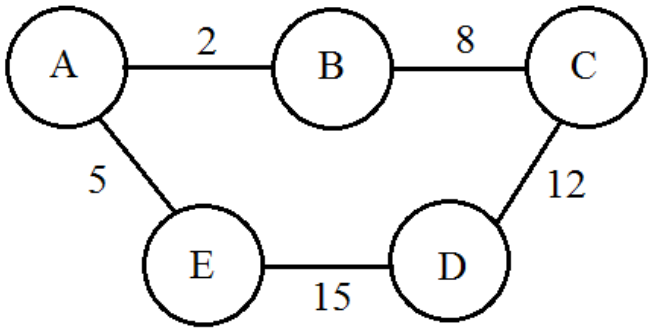


Figure 3

Can this new problem be solved similarly?

[2] Adapted from Numbers in Circles by John Parker and published in Mathematics in School November 2005.

Input

The first line of input will be the number of input sets. Each input set will appear on its own line. The first character on a line will be a '+' or 'x' indicating whether it is an addition or a multiplication problem. Next there will be an odd integer N ≥ 3, followed by N positive integers corresponding to the sum/product of the values in consecutive circles.

Output

For each input set, output one line that contains the N non-negative integers (ordered so that the first pair has the first result listed in the input set, the second and third values have the second result, etc.). If no solution exists for non-negative integers, output "no solution" instead.

Sample Input

```
3

+ 3 11 15 10

x 5 2 8 12 15 5

+ 3 9 1 2
```

Sample Output

```
3 8 7

1 2 4 3 5

no solution
```

Palindromic Sum (prob7)

Sequences of numbers fascinate people. Consider a sequence that starts with a single positive integer. Generate the next number in the sequence by adding the first to the number that is obtained by reversing the digits of the first. This process will continue until a value is generated that is palindromic (the digits are same from right to left as from left to right).
Consider the sequence that starts with 184. The next number in the sequence will be 665 (which is 184 + 481). The next value is 1231 (665 + 566), and the one after that is 2552 (1231 + 1321), a palindrome.
It has been conjectured that every integer will eventually produce a palindrome.
What about 196?
After 2,415,836 reversals and additions, 196 grows to a number of 1,000,000 digits without ever yielding a palindrome. Does it ever produce one? Still, nobody knows. From a probabilistic standpoint, as a number grows to enormous size the likelihood of producing a palindrome on the next step decreases.[3] But with infinite repetition of this process…perhaps.
You will write a program to explore palindromic sums.

Input

The input consists of one or more data sets. Each data set consists of a pair of integers, a non-negative integer representing the initial value and a positive integer representing the maximum number of repetitions.
The end of input is indicated by end of file.

Output

The output for each data set should be a single line giving the initial value, a colon, one space, and either the first palindrome in the sequence and the number of iterations it took to reach the palindrome or a message that no palindrome was found in the given number of iterations.

[3] For more details on this, see http://www.fourmilab.ch/documents/threeyears/threeyears.html

Sample Input

```
184 1
184  10
700 10
999999527 2000
196 3000
```

Sample Output

```
184: no palindrome found in 1 repetitions
184: 2552 (3 repetitions)
700: 707 (1 repetitions)
999999527: 222757757222 (6 repetitions)
196: no palindrome found in 3000 repetitions
```

Please, Let Me Count the Ways (prob8)

A Polite Number is one which can be represented as a sum of two or more consecutive integers. For example, 15 is a Polite Number. In fact, it can be written as the sum of 2 or more integers in 3 different ways:

15 = 1 + 2 + 3 + 4 + 5
15 = 4 + 5 + 6
15 = 7 + 8

The number of ways a Polite Number can be written is equal to the number of odd factors greater than 1 it has. A quick check shows that 15 has 3 odd factors greater than 1: 3, 5, and 15.

For this program, you will give all of the ways numbers can be represented as sums of consecutive, positive integers.

Input

The input will consist of 1 or more test case. Each test case is a single integer n, $0 < n < 1{,}000{,}000$.

The end of input is marked by a test case with the value 0, which should not be processed.

Output

For each test case, either give all of the ways the value can be represented as the sum of consecutive positive, integers, giving just the first and last integer in the sequence. If there is more than one sequence, print one per line and give the sequences in order of their lower values. If there is no such sequence, give the number followed by the message “is not polite”.

Sample Input

```
15

8

27

0
```

Sample Output

```
15 = 1 to 5

15 = 4 to 6

15 = 7 to 8

8 is not polite

27 = 2 to 7

27 = 8 to 10

27 = 13 to 14
```

Prefix Free Set (prob9)

You want to build some strings and you are now in a shop which sells two upper case letters A and B only. Each A costs 1 and each B costs 4. The cost of a string is just the sum of the cost of its letters. For example AAB will cost 6.
You want to build a set of N strings where none of the string is a prefix of another string in the set. There are many ways to do this, but you want to minimize the total cost of such sets. For example if N is 3 then one possible set is { AAA, B, ABBA} which costs 17, but an optimal set is {AA, AB, B} which costs 11.

Input

First line of the input contains *T*, a positive integer representing the number of test cases. The next *T* lines each hold one test case. Each test case contains an integer *N* ($1 \le N \le 10^9$) denoting the length of the desired prefix free set.

Output

For each test case output the size of the test case followed by a colon and the minimum cost of a prefix free set of size N which uses only the letters A and B.

Sample Input

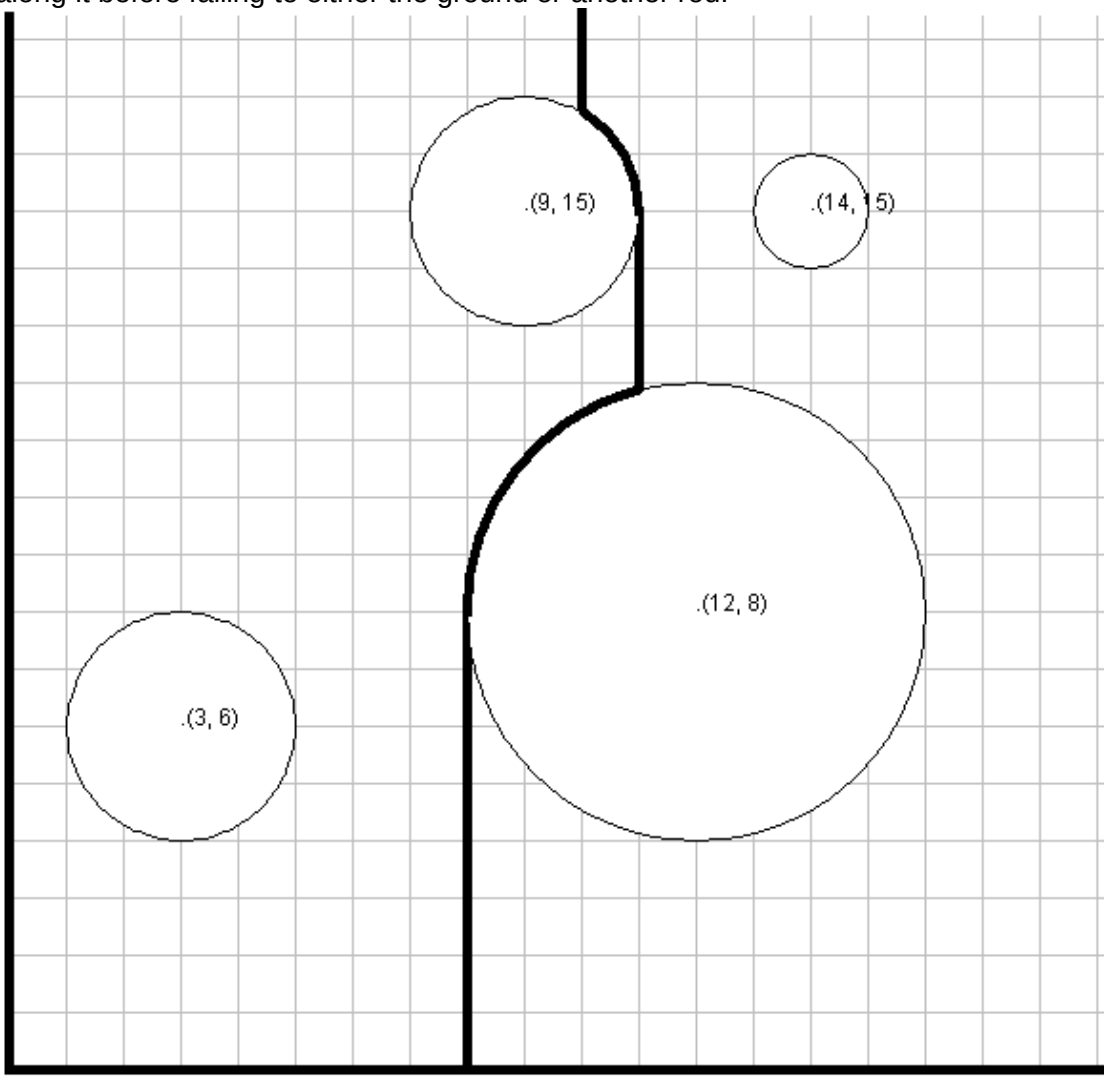
```
5
1
10
100
1000
10000
```

Sample Output

```
1: 1
10: 74
100: 1455
1000: 21708
10000: 288578
```

Raindrops (prob10)

There is a world filled with rods of varying thicknesses. A raindrop falls from above these rods and either falls directly to the ground (if there is no rod directly below it) or hit a rod and runs along it before falling to either the ground or another rod.



Your program will take a configuration of rods and will determine the total distance a raindrop will fall, given its starting position.

Input

The input to this program will consist of one or more test sets.
Each test set will begin with a pair of integers, *rods* and *drops*, 0 ≤ *rods* ≤ 100 and 0 ≤ *drops* ≤ 100. There will then be *rods* lines, each defining one rod. These lines will consist of three integers: *center-x*, *center-y*, *radius*, 0 ≤ *center-x, center-y* ≤ 1000 and 1 ≤ *radius* ≤ 100, where *center-x* and *center-y* represent the center of the rod and *radius* represents its radius. No rod will touch the ground and no two rods will touch each other (treat rods as solid disks so a rod will not be inside another rod) .
There will then be *drops* lines, each with a single integer representing the x position of the raindrop at the start. All drops should be assumed to start at height 1200.
The end of input is represented by a test set with *drops* = 0.

Output

For each test set, have a line with the number of the test set, starting at 1. Then, for each drop, give the total distance the drop falls before reaching the ground, rounded to 4 decimal places. Obviously, drops never fall upward. In the case a drop falls on the highest point of a rod, it will move to the left (smaller x) side of the rod.

Sample Input

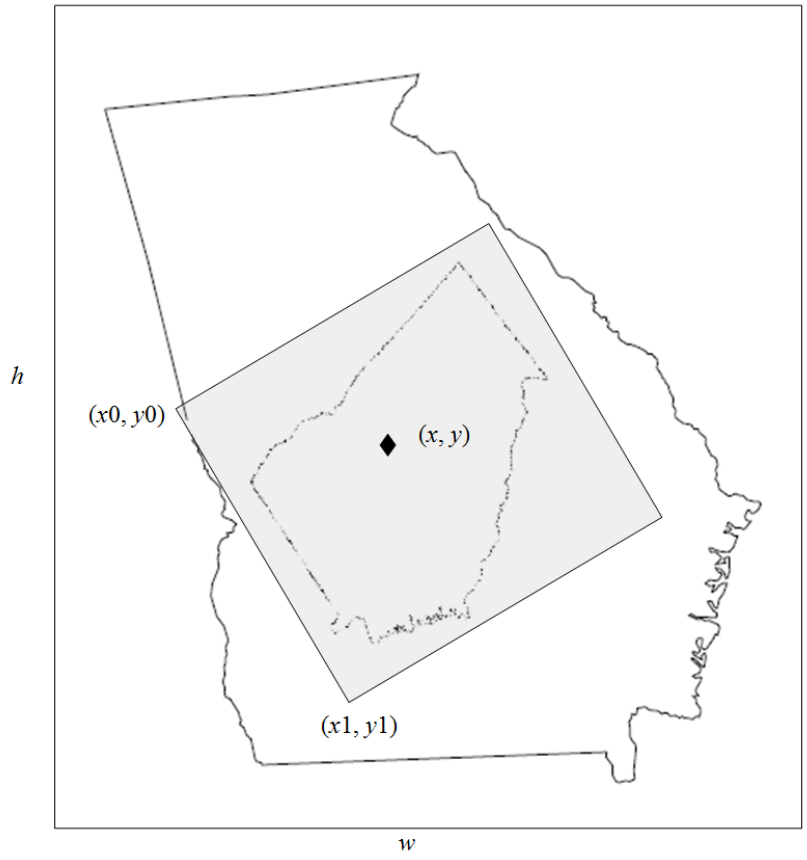
```
4 1
3 6 2
9 15 2
12 8 4
14 15 1
10
0 2
17
19
3 0
```

Sample Output

```
Test set 1

1200.1975 1201.7618

Test set 2

1200.0000

1200.0000
```

Treasure Maps (prob11)

Captain Flint made two treasure maps to mark the location of his buried treasure. Except for the different scales, the two maps were identical. They had similar rectangular shapes and represented exactly the same geographical region, but one map was strictly larger than the other. Captain Flint carefully placed the smaller map on top of the larger map in such a way that the location of the treasure on one map coincided with the location of the treasure on the other map. Parts of the small map may extend past the edge of the larger map. He then put a pin at this point so that the pin pointed to the location of the buried treasure on both maps.
When you discovered the treasure maps two centuries later, unfortunately, the pin was long gone and there were many additional holes in each map, but the maps seemed to be in their original positions. You realized that the secret location could be calculated precisely from the positions of the two maps, because it must be the unique point where both maps point to the same location.



The larger map of size *w* x *h* is aligned with the *xy*-coordinate axes and has its lower-left corner at the origin (0,0). The smaller map has its corresponding lower-left corner at (*x*0, *y*0) and lower-right corner at (*x*1, *y*1).

Input

There will be several sets of input. Each set will consist of six real numbers on a single line delimited by a single space:

w h x0 y0 x1 y1

The dimension of the larger map is *w* x *h*. The lower-left corner and lower-right corner of the smaller map have coordinates (*x*0, *y*0) and (*x*1, *y*1), respectively.
The end of input is marked by end of file.

Output

For each input set, print the coordinates of the location of the buried treasure in a line. The coordinates should be printed as two real numbers delimited by a single space. Print all values with 6 decimal places of precision (rounded). If no such point common to both maps exists, print the text "`Can't find the treasure`" in a single line. There should be no blank lines between outputs.

Sample Input

```
10 10 5 0 10 5

10 10 5 0 5 5

10 10 5 5 10.1 5

20 40 -5 5 5 -5
```

Sample Output

```
5.000000 5.000000

4.000000 2.000000

Can't find the treasure

0.000000 10.000000
```

Who Wants to be a Code for America City? (prob12)

Cities are applying now to be hosts for Code for America in 2013. There is a list of requirements for host cities at codeforamerica.org, but all cities must have leadership, flexibility, involvement, and planning to be considered. For this program, you will read a group of proposals from cities and indicate which meet these minimum standards.

Input

The input will consist of one or more data sets.
The first line of each set will be a string of 1 or more characters with the name of the city. The next 1 or more lines will consist of text which is the application from the given city. The end of input for each data set will be a blank line (a line with just whitespace).
The end of input is marked by a city named End. This line should not be processed.

Output

For each city, have a single line of output with the name of the city, followed by a colon and then one of the phrases “Meets standards” or “Does not meet standards” .
A city will meet the standards if the strings “lead” “flexib” “involve” and “plan” appear in any case in any order in the text of city’s application. These strings will not be broken across a line, but they may overlap each other. A city will not meet standards if it does not have all 4 strings.

Sample Input

```
Macon, GA

We’re glad you’re here in our vibrant, strong, yet flexible city.
We’re leading the
way to a better tomorrow.  We’re glad you want to be involved.
Plan on staying longer the next time you’re here.



Chicken, Alaska
We’re pleading with you!  Despite the inflexibility of our
government, we need help.
Be involved, hop on a plane, and come be Chicken.

Confidence, California, USA
Sure, we’d love to have you and can pay lots of money if you’ll be
involved in our master city plan.

San Francisco, CA

Its all about knowing how to write a proposal.
fLexiBleADInVoLvEPLan.

End
```

Sample Output

```
Macon, GA: Meets standards

Chicken, Alaska: Meets standards

Confidence, California, USA: Does not meet standards

San Francisco, CA: Meets standards
```