MIT Programming Contest
Team Contest Problems
2003

October 5th, 2003

1 XYZZY

It has recently been discovered how to run open-source software on the Y-Crate gaming device. A number of enterprising designers have developed Advent-style games for deployment on the Y-Crate. Your job is to test a number of these designs to see which are winnable.

Each game consists of a set of up to 100 rooms. One of the rooms is the start and one of the rooms is the finish. Each room has an energy value between -100 and +100. One-way doorways interconnect pairs of rooms.

The player begins in the start room with 100 energy points. She may pass through any doorway that connects the room she is in to another room, thus entering the other room. The energy value of this room is added to the player's energy. This process continues until she wins by entering the finish room or dies by running out of energy (or quits in frustration). During her adventure the player may enter the same room several times, receiving its energy each time.

Input

The input consists of several test cases. Each test case begins with n, the number of rooms. The rooms are numbered from 1 (the start room) to n (the finish room). Input for the n rooms follows. The input for each room consists of one or more lines containing:

- the energy value for room i;
- the number of doorways leaving room i;
- a list of the rooms that are reachable by the doorways leaving room i.

The start and finish rooms will always have energy level 0. A line containing -1 follows the last test case.

Output

In one line for each case, output ”winnable” if it is possible for the player to win, otherwise output ”hopeless”.

Example

Sample Input

```
5
0 1 2
-60 1 3
-60 1 4
20 1 5
0 0
5
0 1 2
20 1 3
-60 1 4
-60 1 5
0 0
5
0 1 2
21 1 3
-60 1 4
-60 1 5
0 0
5
0 1 2
20 2 1 3
-60 1 4
-60 1 5
0 0
-1
```

Sample Output

```
hopeless
hopeless
winnable
winnable
```

2 Catenyms

A catenym is a pair of words separated by a period such that the last letter of the first word is the same as the last first of the second. For example, the following are catenyms:

```
dog.gopher
gopher.rat
rat.tiger
aloha.aloha
arachnid.dog
```

A compound catenym is a sequence of three or more words separated by periods such that each adjacent pair of words forms a catenym. For example,

```
aloha.aloha.arachnid.dog.gopher.rat.tiger
```

Given a dictionary of lower case words, you are to find a compound catenym that contains each of the words exactly once.

Input

The first line of standard input contains t, the number of test cases. Each test case begins with $3 \leq n \leq 1000$ - the number of words in the dictionary. n distinct dictionary words follow; each word is a string of between 1 and 20 lowercase letters on a line by itself.

Output

For each test case, output a line giving the lexicographically least compound catenym that contains each dictionary word exactly once. Output "***" if there is no solution.

Example

Sample Input

```
2
6
aloha
arachnid
dog
gopher
rat
tiger
3
oak
maple
elm
```

Sample Output

```
aloha.arachnid.dog.gopher.rat.tiger
***
```

This page intentionally left blank.

3 A Brief Gerrymander

The evil ruling party, the Liberatives, are redistributing the electoral regions (ridings) in your city, and are nefariously attempting to pack certain opposition-friendly neighborhoods into as few ridings as possible. If this plan succeeds, it will be the end of democracy as we know it! As a card-carrying member of the noble, compassionate Conservals (the main opposition party), you must foil their devious plot and safeguard the future of our country.

Under the new plan, the city will be divided up into a grid of rectangles by selecting certain major streets and avenues as boundaries. All streets and avenues stretch straight across the city, and are numbered starting from the southwest corner of the city. The city is bounded by four roads: 1st Street (west edge), 100th Street (east edge), 1st Avenue (south edge), 100th Avenue (north edge). Clearly these four roads must represent district boundaries; however, only a subset of the streets and avenues in between will divide districts. The Liberatives have already been able to fix the North-South (street) boundaries; however, they have been forced to allow the Conservals to select the East-West (avenue) boundaries.

You know the location of all the opposition-friendly neighborhoods, which vote strongly Conserval. A neighborhood is exactly one block between adjacent streets and avenues. For instance, one neighborhood might lie between 47th and 48th Street and 67th and 68th Avenue. Place the avenue boundaries so that as many ridings as possible contain at least one opposition-friendly neighborhood.

Input

Input consists of multiple cases, each describing a city. The first line will give N, the number of Conserval neighborhoods in the city; the following N lines will contain the street and avenue number of the southwest corner of each neighborhood. The next line will contain S, the number of street boundaries, and the S street numbers that they lie on, in increasing order. The final line will consist of A (at least 2), the number of avenue boundaries you must place. Input will be terminated by a line containing -1.

Output

For each case, output a single line giving A followed by A distinct avenue numbers for the optimal redistribution, in increasing order. If there are multiple such solutions, any one will do.

Example

Sample Input

```
2
49 49
50 50
2 1 100
3
-1
```

Sample Output

```
3 1 50 100
```

4 Biometrics

Recently, the term Biometrics been used to refer to the emerging field of technology devoted to identification of individuals using biological traits, such as those based on retinal or iris scanning, fingerprints, or face recognition.

A simple biometric system translates a human image into a polygon by considering certain features (eyes, nose, ears, etc.) to be vertices and connecting them with line segments. The polygon has distinct vertices but may be degenerate in that the line segments could intersect. Because these polygons are generally created from remote images, there is some uncertainty as to their scale and rotation. Your job is to determine whether or not two polygons are similar; that is, can they be made equal by repositioning, rotating and magnifying them?

Input

Input consists of several test cases. Each test case consists of three lines containing:

- f, the number of features;
- f coordinate pairs giving the vertices of the first polygon;
- f coordinate pairs giving the vertices of the second polygon.

The vertices for both polygons correspond to the same set of features in the same order; for example, right ear tip, chin cleft, right eye, nose, left eye, left ear tip, space between front teeth. Each polygon has f distinct vertices; each vertex is given as an x and y coordinate pair. There are at least three and no more than ten features. Coordinates are integers between -1000 and 1000. A line containing 0 follows the last test case.

Output

For each case, output a line "similar" or "dissimilar" as appropriate. The two polygons are similar if, after some combination of translation, rotation, and scaling (but not reflection) both vertices corresponding to each feature are in the same position.

Example

Sample Input

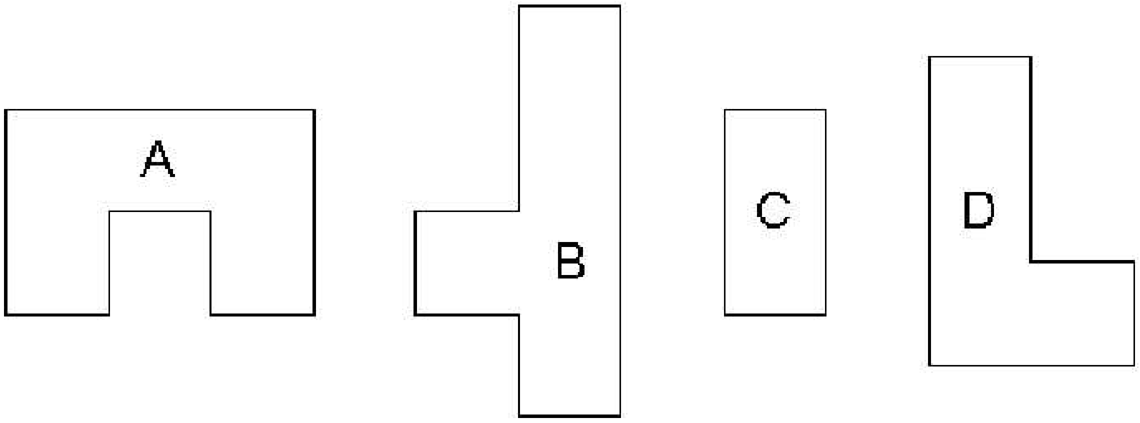
```
4
0 0 0 1 1 1 1 0
0 1 1 0 0 -1 -1 0
3
0 0 10 0 10 10
0 0 -10 0 -10 10
3
0 0 10 10 20 20
0 0 11 11 22 22
3
0 0 10 10 20 20
0 0 11 11 20 20
0
```

Sample Output

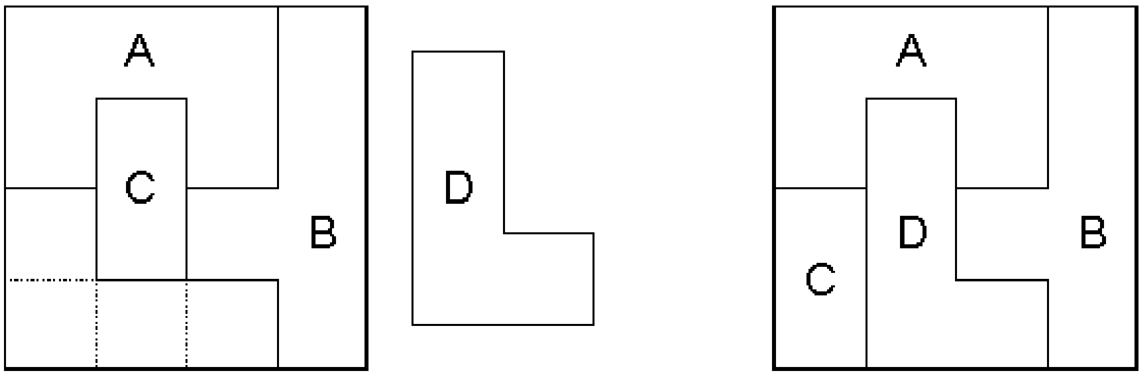
```
similar
dissimilar
similar
dissimilar
```

5 A Puzzling Problem

The goal of this problem is to write a program which will take from 1 to 5 puzzle pieces such as those shown below and arrange them, if possible, to form a square. An example set of pieces is shown here.



The pieces cannot be rotated or flipped from their original orientation in an attempt to form a square from the set. All of the pieces must be used to form the square. There may be more than one possible solution for a set of pieces, and not every arrangement will work even with a set for which a solution can be found. Examples using the above set of pieces are shown here.



Input

The input file for this program contains several puzzles (i.e. sets of puzzle pieces) to be solved. The first line of the file is the number of pieces in the first puzzle. Each piece is then specified by listing a single line with two integers, the number of rows and columns in the piece, followed by one or more lines which specify the shape of the piece. The shape specification consists of '0' and '1' characters, with the '1' characters indicating the solid shape of the puzzle (the '0' characters are merely placeholders). For example, piece 'A' above would be specified as follows:

```
2 3
111
101
```

The pieces should be numbered by the order they are encountered in the puzzle. That is, the first piece in a puzzle is piece #1, the next is piece #2, etc. All pieces may be assumed to be valid and no larger than 4 rows by 4 columns.

The line following the final line of the last piece contains the number of pieces in the next puzzle, again followed by the puzzle pieces and so on. The end of the input file is indicated by a zero in place of the number of puzzle pieces.

Output

Your program should report a solution, if one is possible, in the format shown by the examples below. A 4-row by 4-column square should be created, with each piece occupying its location in the solution. The solid portions of piece #1 should be replaced with '1' characters, of piece #2 with '2' characters, etc. The solutions for each puzzle should be separated by a single blank line. For puzzles which have no possible solution simply report "No solution possible".

Example

Sample Input

```
4
2 3
111
101
4 2
01
01
11
01
2 1
1
1
3 2
10
10
11
4
1 4
1111
1 4
1111
1 4
1111
2 3
111
001
5
2 2
11
11
2 3
111
100
3 2
11
01
01
1 3
111
1 1
1
0
```

Sample Output

```
1112
1412
3422
3442

No solution possible

1133
1153
2223
2444
```

This page intentionally left blank.

6 Cities

In the days of yore, Han was a prosperous kingdom. However, at the turn of the century, hard times fell upon the kingdom. A plague swept through the land, and barbarians galloped in from the north, burned farms, destroyed roads, and pillaged villages. All that remained were a few isolated strongholds scattered throughout the land. It has now been nearly a decade since the last wave of barbarians stormed through and everything has calmed down considerably. People are now becoming revitalized with the hope that they can once again transform the kingdom back into its former glory.

Since all the roads were destroyed, the strongholds are left in isolation, so the first order of business should be to build a network of roads connecting all the strongholds. Each stronghold thought it would be a reasonable plan to start by building a road to the closest stronghold near it. If there are two strongholds of equal distance, they will choose the stronghold whose name comes before the other in the dictionary. Each stronghold is able to build a road at some rate, measured in feet/hour. Because they wish to finish as soon as possible, construction happens 24 hours a day continuously, advancing the construction site (the end of the road) towards the destination. Of course, construction on a road stops if the road runs into another road or a city. At the beginning of the new year, there was a big celebration and all the strongholds began construction at the same time.

Little did the people of Han know that the barbarians had infiltrated their kingdom and were observing the progress of the roads. The barbarians are curious about the progress of the roads. In particular, they want answers to two type of questions.

1. After exactly t hours since construction began, what is the absolute minimum length of additional roads that still need to be built in order to connect all cities? These additional roads are allowed to join two cities, two construction sites, or one of each.

2. What is the fewest number of hours that must elapse before the minimum length of additional roads that still need to be built is at most l?

Write a program to answer these questions given Han's construction plan.

Input

There will be several test cases, each representing a possible scenario for Han. The first line of each test case will contain a positive integer number N, the number of strongholds ($1 \leq N \leq 2000$). Each of the subsequent N lines will contain a description of a stronghold: a name consisting of letters 'a'...'z', the x and y coordinates of the position of the stronghold (in feet), and the construction rate (in feet/hour). The next lines will contain questions. The first integer on the line is either 1 or 2, representing the type of question. For type 1 questions, the next number is t, the time (in hours) in question. For type 2 questions, the next number is l, the length (in feet) in question. The questions will be terminated by a line with 0.

The input data is terminated by a line that contains one zero, and should not be processed.

Output

For each test case, output the answers to each question, formatted as in the sample output. If for question 2, at no point in time will there be l feet of construction left, then print "NEVER." All numeric answers should be printed to as many decimal places as you feel necessary. To get credit for the problem, however, your answer must be within 0.01 of our answer.

Example

Sample Input

```
4
portland 0 0 3
seattle 0 10 2
newyork 20 6 1
boston 20 0 1
1 0
1 2.0
1 3.0
2 29
2 1.0
0
2
bree -10 -10 1
buckland 10 10 2
1 5
0
0
```

Sample Output

```
Kingdom 1
36.000 feet left at time 0.000
22.000 feet left at time 2.000
20.000 feet left at time 3.000
1.000 hours before 29.000 feet left
NEVER

Kingdom 2
13.284 feet left at time 5.000

End
```

7 Calories From Fat

Fat contains about 9 Calories/g of food energy. Protein, sugar, and starch contain about 4 Calories/g, while alcohol contains about 7 Calories/g. Although many people consume more than 50% of their total Calories as fat, most dieticians recommend that this proportion should be 30% or less. For example, by examining a Nutrition Facts label, we see that 3g of fat is 5% of the recommended daily intake based on a 2,000 calorie diet. A quick calculation reveals that the recommended daily intake of fat is therefore 60g; that is, 540 Calories or 27% Calories from fat.

Others recommend radically different amounts of fat. Dean Ornish, for example, suggests that less than 10% of total caloric intake should be fat. On the other hand, Robert Atkins recommends the elimination of all carbohydrate with no restriction on fat. It has been estimated that the average Atkins dieter consumes 61% of Calories from fat.

From a record of food eaten in one day, you are to compute the percent Calories from fat. The record consists of one line of input per food item, giving the quantity of fat, protein, sugar, starch and alcohol in each. Each quantity is an integer followed by a unit, which will be one of: g (grams), C (Calories), or % (percent Calories). Percentages will be between 0 and 99. At least one of the ingredients will be given as a non-zero quantity of grams or Calories (not percent Calories).

Input

Input will consist of several test cases. Each test case will have one or more lines as described above. Each test case will be terminated by a line containing '-'. An additional line containing '-' will follow the last test case.

Output

For each test case, print percent Calories from fat, rounded to the nearest integer.

Example

Sample Input

```
3g 10g 10% 0g 0g
55% 100C 0% 0g 30g
-
25g 0g 0g 0g 0g
-
1g 15% 20% 30% 1C
-
-
```

Sample Output

```
53%
100%
32%
```

This page intentionally left blank.

8 Dead Fraction

Mike is frantically scrambling to finish his thesis at the last minute. He needs to assemble all his research notes into vaguely coherent form in the next 3 days. Unfortunately, he notices that he had been extremely sloppy in his calculations. Whenever he needed to perform arithmetic, he just plugged it into a calculator and scribbled down as much of the answer as he felt was relevant. Whenever a repeating fraction was displayed, Mike simply recorded the first few digits followed by "...". For instance, instead of "1/3" he might have written down "0.3333...". Unfortunately, his results require exact fractions! He doesn't have time to redo every calculation, so he needs you to write a program (and FAST!) to automatically deduce the original fractions.

To make this tenable, he assumes that the original fraction is always the simplest one that produces the given sequence of digits; by simplest, he means the the one with smallest denominator. Also, he assumes that he did not neglect to write down important digits; no digit from the repeating portion of the decimal expansion was left unrecorded (even if this repeating portion was all zeroes).

Input

There are several test cases. For each test case there is one line of input of the form "0.dddd..." where dddd is a string of 1 to 9 digits, not all zero. A line containing 0 follows the last case.

Output

For each case, output the original fraction. Note that an exact decimal fraction has two repeating expansions (e.g. 1/5 = 0.2000... = 0.19999...).

Example

Sample Input	Sample Output
`0.2...`	`2/9`
`0.20...`	`1/5`
`0.474612399...`	`1186531/2500000`
`0`	

This page intentionally left blank.