2006 Mid-Atlantic Regional Programming Contest

Welcome to the the 2006 Programming Contest. Before you start the contest, please be aware of the following notes:

1. There are eight (8) problems in the packet, using letters A–H. These problems are NOT sorted by difficulty. As a team's solution is judged correct, the team will be awarded a balloon. The balloon colors are as follows:

Problem	Problem Name	Balloon Color
A	Permutation Recovery	Gold
B	Crosswords Insider	Orange
C	Doors and Penguins	Purple
D	Gypsy Moths	Silver
E	Marbles in Three Baskets	Blue
F	Generic Units Converter	Pink
G	Zoned Out	Green
H	Shrew-ology	Red

2. All solutions must read from standard input and write to standard output. In C this is scanf/printf, in C++ this is cin/cout, and in Java this is System.in/System.out. The judges will ignore all output sent to standard error. (You may wish to use standard error to output debugging information.) From your workstation you may test your program with an input file by redirecting input from a file:

   ```
   program < file.in
   ```

3. Solutions for problems submitted for judging are called runs. Each run will be judged.

   The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
Correct	Your submission has been judged correct.
Wrong Answer	Your submission generated output that is not correct or is incomplete.
Output Format Error	Your submission's output is not in the correct format or is misspelled.
Excessive Output	Your submission generated output in addition to or instead of what is required.
Compilation Error	Your submission failed to compile.
Run-Time Error	Your submission experienced a run-time error.
Time-Limit Exceeded	Your submission did not solve the judges' test data within 30 seconds.

4. A team's score is based on the number of problems they solve and penalty points, which reflect the amount of time and number of incorrect submissions made before the problem is solved. For each problem solved correctly, penalty points are charged equal to the time at which the problem was solved plus 20 minutes for each incorrect submission. No penalty points are added for problems that are never solved. Teams are ranked first by the number of problems solved and then by the fewest penalty points.

5. This problem set contains sample input and output for each problem. However, you may be assured that the judges will test your submission against several other more complex datasets, which will not be revealed until after the contest. Your major challenge is designing other input sets for yourself so that you may fully test your program before submitting your run. Should you receive an incorrect judgment, you should consider what other datasets you could design to further evaluate your program.

6. In the event that you feel a problem statement is ambiguous or incorrect, you may request a clarification. Read the problem carefully before requesting a clarification. If the judges believe that the problem statement is sufficiently clear, you will receive the response, "The problem statement is sufficient; no clarification is necessary." If you receive this response, you should read the problem description more carefully. If you still feel there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found. If the problem statement is ambiguous in specifying the correct output for a particular input, please include that input data in the clarification request.

   You may not submit clarification request asking for the correct output for inputs that you provide.[1] Sample inputs *may* be useful in explaining the nature of a perceived ambiguity, e.g., "The is no statement about the desired order of outputs. Given the input: ..., would not both this: ... and this: ... be valid outputs?".

   If a clarification is issued during the contest, it will be broadcast to all teams.

7. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for B followed by a run for C, but receive the response for C first.

   Do not request clarifications on when a response will be returned. If you have not received a response for a run within 30 minutes of submitting it, **you may have a runner ask the site judge to determine the cause of the delay. Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.**

   If, due to unforeseen circumstances, judging for one or more problems begins to lag more than 30 minutes behind submissions, a clarification announcement will be issued to all teams. This announcement will include a change to the 30 minute time period that teams are expected to wait before consulting the site judge.

[1] Note that is a change to the rules used for this contest in recent years. It brings us into conformance with the practices followed by most other regional contests and in the international final.

8. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification.

9. All lines of program input and output should end with a newline character (`\n`, `endl`, or `println()`).

10. Unless otherwise specified, all lines of program output should be left justified, with no leading blank spaces prior to the first non-blank character on that line.

11. All input sets used by the judges will follow the input format specification found in the problem description.

12. Unless otherwise specified, all numbers will appear in the input and should be presented in the output beginning with the − if negative, followed immediately by 1 or more decimal digits. If it is a real number, then the decimal point appears, followed by any number of decimal digits (for output of real numbers the number of decimal digits will be specified in the problem description as the "precision"). All real numbers printed to a given precision should be rounded to the nearest value.

    In simpler terms, neither scientific notation nor commas will be used for numbers, and you should ensure you use a printing technique that rounds to the appropriate precision.

13. Good luck, and HAVE FUN!!!

Problem A: Permutation Recovery

Professor Permula gave a number of permutations of the n integers $1, 2, \ldots, n$ to her students. For each integer i, ($1 \leq i \leq n$), she asks the students to write down the number of integers greater than i that appear before i in the given permutation. This number is denoted a_i. For example, if $n = 8$ and the permutation is $2, 7, 3, 5, 4, 1, 8, 6$, then $a_1 = 5$ because there are 5 numbers $(2, 7, 3, 5, 4)$ greater than 1 appearing before it. Similarly, $a_4 = 2$ because there are 2 numbers $(7, 5)$ greater than 4 appearing before it.

John, one of the students in the class, is studying for the final exams now. He found out that he has lost the assignment questions. He only has the answers (the a_i's) but not the original permutation. Can you help him determine the original permutation, so he can review how to obtain the answers?

Input

The input consists of a number of test cases. Each test case starts with a line containing the integer `n` ($\texttt{n} \leq 500$). The next `n` lines give the values of $a_1, a_2, \ldots, a_n$. The input ends with `n` = 0.

Output

For each test case, print a line specifying the original permutation. Adjacent elements of a permutation should be separated by a comma. Note that some cases may require you to print lines containing more than 80 characters.

Example

Input

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

```
2
1
0
0
```

Output:

```
2,7,3,5,4,1,8,6
10,9,8,7,6,5,4,3,2,1
```

Problem B: Crosswords Insider

You have an insider at the New York Times who sends you a list of the answers for the crossword puzzle, but his list does not say which answer goes with which clue. His list occasionally contains errors or omissions, but is usually correct. Given the list of answers and the shape of the crossword puzzle, figure out a valid assignment.

Input

The input will consist of one or more problem sets.

Each problem set begins with a line containing two integers `M` and `N`. Zero values for these will indicate the end of the problem sets. `M` denotes the number of words to be placed into the puzzle and will be in the range $1 \ldots 150$. `N` denotes the number of rows in the puzzle and will be in the range $1 \ldots 16$.

This is followed by `M` lines, each containing a single word, left-justified on the line. A word will contain only alphabetic characters and words will not be duplicated within any problem set. Words will be $2 \ldots 16$ characters in length.

This is followed by `N` lines denoting a puzzle template as a series of '.' and '#' characters, left-justified and followed immediately by the end of line. Each line will contain the same number of these characters. That number may range from $1 \ldots 16$. A '.' indicates a position in the puzzle where a character may be written. A '#' indicates a position at which a character may not appear.

Output

Your program should determine if all the given words can be arranged into the puzzle template in such a way as to leave no '.' positions unfilled and without filling any '#' positions. Vertical and horizontal words may intersect at a common letter, but two horizontal or two vertical words may neither intersect nor be adjacent to one another without at least one intervening '#'.

Your program should print the string "Problem " followed by the problem set number. If no solution is possible, it should then print, on the remainder of that output line, the string ": No layout is possible.".

If a solution is possible, then the program should, beginning on the next line after the problem set number, print the `N` lines of the crossword puzzle as presented in the input but with the appropriate characters substituted for the '.' positions.

If multiple solutions are possible, you may print any solution.

Example

Input

```
4 4
tow
cat
row
care
```

```
...
.#.
...
.##
0 0
```

Output:

```
Problem 1
cat
a#o
row
e##
Problem 2: no layout is possible.
```

Problem C: Doors and Penguins

The organizers of the Annual Computing Meeting have invited a number of vendors to set up booths in a large exhibition hall during the meeting to showcase their latest products. As the vendors set up their booths at their assigned locations, they discovered that the organizers did not take into account an important fact — each vendor supports either the Doors operating system or the Penguin operating system, but not both. A vendor supporting one operating system does not want a booth next to one supporting another operating system.

Unfortunately the booths have already been assigned and even set up. There is no time to reassign the booths or have them moved. To make matter worse, these vendors in fact do not even want to be in the same room with vendors supporting a different operating system.

Luckily, the organizers found some portable partition screens to build a wall that can separate the two groups of vendors. They have enough material to build a wall of any length. The screens can only be used to build a straight wall. The organizers need your help to determine if it is possible to separate the two groups of vendors by a single straight wall built from the portable screens. The wall built must not touch any vendor booth (but it may be arbitrarily close to touching a booth). This will hopefully prevent one of the vendors from knocking the wall over accidentally.

Input

The input consists of a number of cases. Each case starts with 2 integers on a line separated by a single space: D and P, the number of vendors supporting the Doors and Penguins operating system, respectively ($1 \leq D, P \leq 500$). The next D lines specify the locations of the vendors supporting Doors. This is followed by P lines specifying the locations of the vendors supporting Penguins. The location of each vendor is specified by four positive integers: x_1, y_1, x_2, y_2. (x_1, y_1) specifies the coordinates of the southwest corner of the booth while (x_2, y_2) specifies the coordinates of the northeast corner. The coordinates satisfy $x_1 < x_2$ and $y_1 < y_2$. All booths are rectangular and have sides parallel to one of the compass directions. The coordinates of the southwest corner of the exhibition hall is $(0, 0)$ and the coordinates of the northeast corner is $(15000, 15000)$. You may assume that all vendor booths are completely inside the exhibition hall and do not touch the walls of the hall. The booths do not overlap or touch each other.

The end of input is indicated by $D = P = 0$.

Output

For each case, print the case number (starting from 1), followed by a colon and a space. Next, print the sentence:

```
It is possible to separate the two groups of vendors.
```

if it is possible to do so. Otherwise, print the sentence:

```
It is not possible to separate the two groups of vendors.
```

Print a blank line between consecutive cases.

Example

Input

```
3 3
10 40 20 50
50 80 60 90
30 60 40 70
30 30 40 40
50 50 60 60
10 10 20 20
2 1
10 10 20 20
40 10 50 20
25 12 35 40
0 0
```

Output:

```
Case 1: It is possible to separate the two groups of vendors.

Case 2: It is not possible to separate the two groups of vendors.
```

Problem D: Gypsy Moths

"The number of infestations is getting worse", said the Ranger. "If we don't get those moths under control, there won't be a healthy tree left in the whole park. Let me show you."

He spread out a map, liberally decorated with red X's. "With our remote-control camera atop Mount Hiabove, we can easily spot the infested trees. Each mark on this map shows an infested site."

"Well", said the Park Manager, "we have a congressional delegation visiting tomorrow. I don't think they'll be impressed by a map, but if we could show them the feed from that camera as it pans around..."

"That's a problem", said the Ranger. "The temperature on the mountain is dropping. By the time they get here, the camera will probably be frozen in place. We better not assume we'll be able to move the camera. And, remember, it doesn't have a very wide field of view. It only shows a rather narrow angle."

"We'll have to do the best we can", said the Manager. "Let's point the camera right now in the direction that will show the most infested trees. Let's see...". He began to study the map.

Input

Input consists of one or more data sets.

In each data set, line 1 contains the non-negative number N of infested trees. A value of 0 indicates end of input.

For each data set, line 2 contains the X,Y coordinates of the camera and the field of view of the camera. Lines $3 \ldots N+2$ each contain the X,Y coordinates of an infested tree. None of these will be the same as the coordinates of the camera;

Coordinates are given as floating point numbers in the range $-500.0 \ldots 500.0$. The field of view is given as a floating-point angle, in degrees, in the range $0.1 \ldots 179.9$.

Output

Determine the angle A that maximizes the number of infested trees visible within the angular range $A \pm V/2$ where V is the angular field of view of the camera. The camera positioning system is calibrated in tenths of a degree and "clicks" into place at each tenth of a degree – intermediate values are not possible. A tree is considered "visible" if it lies inside the $A \pm V/2$ - trees that lie exactly on the border are not considered visible.[2]

The angle 0.0 corresponds to the positive Y axis and the angle 90.0 corresponds to the positive X axis. Assume that distance is not a factor in the visibility of trees.

If more than one angle allows viewing of the same maximal number of infestations, choose the smallest such angle.

For each input set, produce a single line of output of the form:

```
Point the camera at angle ### to view ## infested trees.
```

[2]For contest purposes, judging data will not rely upon calculations resolving trees' locations to within a finer precision than the closest 0.01 degree.

The angle should be printed in degrees in the range $0.0 \ldots 359.9$ with one digit after the decimal. The number of visible infested trees should be printed as an integer.

Example

Input

```
5
25.0 25.0 45.0
25.5 35.0
35.0 30.0
45.0 45.0
40.0 48.5
20.0 18.0
0
```

Output:

```
Point the camera at angle 22.6 to view 3 infested trees.
```

Problem E: Marbles in Three Baskets

Each of three baskets contains a certain number of marbles. You may move from one basket into another basket as many marbles as are already there, thus doubling the quantity in the basket that received the marbles. You must find a sequence of moves that will yield the same number of marbles in the three baskets. Moreover, you must achieve the goal in the smallest possible number of moves. Your program must also recognize the case in which there is no such sequence of moves.

Input

Each line of the input file will contain data for one instance of the problem: three positive integers, with one blank space separating adjacent integers. The three integers represent the initial numbers of marbles in the three baskets. The sum of the three integers will be at most 60.

End of input will be signaled by a line with three zeros.

Output

The output will begin with the initial configuration from the input.

Thereafter, on successive lines, the number of marbles in the respective baskets will be printed after each move, concluding with the line in which the three numbers are identical. As stated above, the goal must be achieved in the smallest possible number of moves. If there is more than one sequence of moves that can achieve the goal in the same minimal number of steps, any such sequence may be printed.

If there is no sequence of moves to achieve the goal, only the initial configuration will be printed.

In all the above lines of output, each integer in the output will be right-justified in a field of width 4.

Each instance of the problem will be concluded by a line of 12 equal signs.

Example

Input

```
6 7 11
15 18 3
5 6 7
0 0 0
```

Output:

```
   6   7  11
   6  14   4
  12   8   4
   8   8   8
============
```

```
  15  18   3
  12  18   6
  12  12  12
============
   5   6   7
============
   0   0   0
============
```

Problem F: Generic Units Conversion

Design a program that can read a description of two systems of measurement for some common quantity (e.g., length, weight, area, time, etc.), a conversion rule that relates the two systems to one another, and a quantity expressed in one measurement system, and that can express that same quantity in the other system.

Input

The input will consist of 1 or more problem sets. Each problem set consists of specifications for two systems of measurement, a conversion rule, and a set of quantities to be converted.

Each problem set begins with a specification of the first system of measurement.

The first line of this specification gives the unit names for the system, presented as a set of one or more words on a single line, separated by single blanks. This line will be at most 80 characters long. Each word on that line names a single unit of measurement and is made up entirely of alphabetic characters. The order of the words will be from the largest unit to the smallest within that system of measurement, and no unit name will be repeated within this line.

This is followed by $N - 1$ lines (where N is the number of units named on the first line) giving internal conversion rules in the form

```
### unit1 = ### unit2
```

where the `###` are positive integer or floating point numbers and "`unit1`" and "`unit2`" are unit names drawn from the first line of the specification. This set of $N - 1$ lines will be well-formed in the sense of providing enough information to convert any unit in the system into an appropriate value in any other unit.

The first system specification is followed immediately by a second, in the same format.

The second specification is followed by a conversion rule, in the same format as the internal conversion rule described above, but "`unit1`" will be drawn from the first system of measurement and "`unit2`" from the second.

The conversion rule is followed by one or more lines, each line giving a quantity in the first system. A quantity is expressed as one or more pairs, each pair consisting of a non-negative number followed by a unit name. Unit names in a quantity will be presented in descending order of size, though not all units in the system will necessarily be mentioned in every quantity.

All specification rules, conversion rules, and quantities will be restricted to ranges for which the desired output (see below) can be contained in normal (not "long") integers.

The end of the list of quantities, and the end of that problem set, is indicated by a completely empty line.

If the next line after that empty one is non-empty, it represents the start of another problem set. If that next line is also empty, however, that indicates the end of the input to this program.

Output

For each input quantity, print a single line giving the equivalent quantity in the second system of measurement. The output will be presented as a series of pairs, each pair a number and a unit

name, with all units in the system included (even if the corresponding number for that unit is zero). The pairs must be presented in decreasing order by unit size and should maximize the value of the larger units over the smaller. All numbers will be integers, and the number for the smallest unit will be rounded to the nearest integral value. Within the output line, numbers and unit names shall be separated from one another by a single space.

Example

Input

```
miles yards feet
5280 feet = 1 miles
3 feet = 1 yards
km m cm
1000 m = 1 km
0.01 m = 1 cm
1 feet = 30.48 cm
2 miles 1 feet
0.0833 feet

furlongs fathoms
1 furlongs = 110 fathoms
feet inches
12 inches = 1 feet
1 fathoms = 6 feet
1 furlongs
0.5 furlongs 0.25 fathoms
```

Output:

```
3 km 218 m 99 cm
0 km 0 m 3 cm
660 feet 0 inches
331 feet 6 inches
```

Problem G: Zoned Out

"You want to build an office building in our town? Wonderful! We're so glad to have you!" The mayor was beaming as he led me back into City Hall. "We just need to make sure your building permit application meets all the town zoning regulations. We can check that immediately."

He led me through a door labelled "Zoning". I was staggered by the sight of a huge room with row after row of desks. At each desk sat a clerk with a logbook, an inbox, and a generaous supply of pencils and erasers.

The mayor took my application form, then flourished a sheet of paper containing several lines of numbered boxes. He placed the paper in the inbox of the nearest desk and said, "We just run this past our zoning clerks, and if it gets back to this desk with all the boxes checked, then your application is approved." The clerk at the desk grabbed the form from the inbox, marked checks in a few boxes, wrote a few lines into a logbook on his desk, then ran to a nearby copier, made copies of the form and distributed to several other desks. I watched as the clerks at those other desks each added more checks but also erased some, wrote in their logbooks, made copies, and distributed the altered forms. Soon copies of the form seemed to be flying all around the room.

I turned to the mayor and said, "But none of them actually *looked* at my application!".

"Oh no", said the Mayor, "each of our clerks has carefully studied a few parts of our zoning regulations. Some are convinced that *any* application will automatically satisfy some of the zoning rules. They also believe that *no* application will ever satisfy some of the other rules. So each clerk simply checks off some boxes and erases the check marks from some others."

"We're very proud", he added, "of the decisive nature of our zoning process. We recruit our staff from the nearby law school and choose only the most officious and argumentative junior students."

"How," I asked, "do they decide where the marked-up copies should go?"

"I'm not really sure," said the mayor. "I think they mainly give them to their fellow clerks that they most dislike, trying to increase the workload of their enemies." Just then a sheet of paper was placed on the closest desk. "Let's see how you're coming along", he said, picking up the paper. "No, still four boxes left empty. We'll just have to wait and see if you can do better."

I sighed and said, "It seems to me that this could go on forever."

"No", said the mayor. "Each clerk keeps a logbook recording all the versions of your application that they have sent on. When they get a form, they start by adding all marks they have ever seen on prior versions of the form, then do their own marks and erasures. If, after all that, it matches one of the previous versions, they won't send it on a second time."

I resigned myself to a long wait.

Input

The input consists of one or more problem sets.

The first line in each set contains the number of boxes on the form and the number of clerks in the office. Both are positive integers. The end of input is signalled by a line with zeros for both of these numbers.

The remainder of the input consists of three lines of data for each clerk, starting with clerk #0. (Boxes and clerks are both identified by number, starting at 0.) Each line will contain 0 or more integers, separated by whitespace, as follows:

- One line contains the numbers of the boxes that always marked by that clerk.
- The next line contains the numbers of the boxes that are always erased by that clerk.
- The third line contains the numbers of all those clerks to whom this one sends copies of the altered form.

Output

For each problem set, print a single line containing the numbers of the boxes marked on the last copy of the form that leaves the desk of clerk #0. The numbers should be printed in ascending order, separated by a single blank space.

Example

Input

```
6 5
1

1 2
2 3
4
4
5
1
3
2

1
4
2
1 0
0 0
```

Output:

```
1 3 4 5
```

Problem H: Shrew-ology

Dr. Montgomery Moreau has been observing a population of Northern Madagascar Pie-bald Shrews in the wild for many years. He has made careful observations of all the shrews in the area, noting their distinctive physical characteristics and naming each one.

He has made a list of significant physical characteristics (e.g., brown fur, red eyes, white feet, prominent incisor teeth, etc.) and taken note of which if these appear to be dominant (if either parent has this characteristic, their children will have it) or recessive (the children have this characteristic only if both parents have it).

Unfortunately, his funding from the International Zoological Institute expired and he was forced to leave the area for several months until he could obtain a new grant. During that time a new generation was born and began to mature. Upon returning, Dr. Moreau hopes to resume his work, starting by determining the likely parentage of the each member of the new generation.

Input

The first line of input will containing a sequence of 1 to 80 consecutive 'D' and 'R' characters describing a list of physical characteristics, indicating whether each is dominant or recessive.

After this line will follow several lines, each describing a single adult shrew. Each shrew is described by a name of 1-32 non-blank characters terminated by a blank space, then a single M or F character indicating the gender of the animal, another blank space, then a list of consecutive 0 or 1 characters, describing the animal. A 1 indicates that the animal possesses that physical characteristic, a 0 indicates that it does not. The list of adults is terminated by a line containing only the string "***".

This is followed by one or more lines describing juvenile animals. These contain a name and description, each formatted identically to those for the adults, separated by a blank space. The list of juveniles is terminated by a line containing only the string "***".

Output

For each juvenile animal, print a single line consisting of the animal's name, the string " by ", then a (possibly empty) list of all possible parents for that animal. A set of parents should be printed as the name of the mother, a hyphen, then the name of the father. If the animal has multiple pairs of possible parents, these pairs should be printed in alphabetic (lexicographic) order first by the mother's name, then by the father's name among pairs where the mother is the same. Each pair should be printed separated by the string " or ".

Example

Input

```
RDDR
Speedy M 0101
Jumper F 0101
Slowpoke M 1101
```

```
Terror F 1100
Shadow F 1001
***
Frisky 0101
Sleepy 1101
***
```

Output:

```
Frisky by Jumper-Slowpoke or Jumper-Speedy or Shadow-Speedy
Sleepy by Shadow-Slowpoke
```