2007 Mid-Atlantic Regional Programming Contest

Welcome to the the 2007 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	First Composed, Then Transposed	Gold
B	Mobiles Alabama	Orange
C	Out of Sight	Purple
D	Witness Redaction Program	Silver
E	Safety in Alchemy	Blue
F	Potholes	Pink
G	Bulls and Cows	Green
H	The Turn of the Shrew	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 requests asking for the correct output for inputs that you provide.Sample inputs *may* be useful in explaining the nature of a perceived ambiguity, e.g., "There 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.

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. All input sets used by the judges will follow the input format specification found in the problem description. You do not need to test for input that violates the input format specified in the problem.

11. If a problem specifies that an input is a floating point number, decimal points may be omitted for numbers with no non-zero decimal portion. Scientific notation will not be used in input sets unless a problem explicitly allows it.

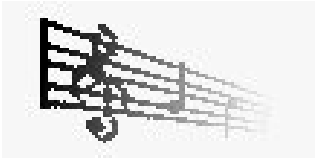
12. Unless otherwise specified, all lines of program output should be left justified, with no leading blank spaces prior to the first non-blank character on that line.

13. 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 digits after the decimal point 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.

14. Good luck, and HAVE FUN!!!

Problem A: First Composed, Then Transposed



The conventional western musical scale is based on 12 tones:
`A A# B C C# D D# E F F# G G#`

The sequence repeats indefinitely - the next note higher than `G#` is also called `A`.

Each step from one note to another in this sequence is called a half-step. The `#` symbol is called a "sharp" and actually means "raised a half step". Hence "`A#`" is "A raised a half-step". There is a similar symbol "flat" meaning "lowered a half-step" that we will render as "`b`" in this problem, though it is normally written as ♭.

Given that meaning for `#` and `b`, it is possible for notes to actually be referred to by more than one name. `A#` and `Bb` refer to the same note. `B#` means the same as `C`. `Fb` means the same as `E`.

A common task for musical arrangers is to transpose a work of music - to raise or lower the entire work by some number of half-steps in order to fit better into the range of a particular singer or musical instrument. A proper transposition will preserve the number of half-steps between any two successive notes in the work.

Input

Input consists of one or more input sets.

An input set consists of a line containing a sequence of 0 or more notes with sharp and flat marks, each note separated fro mthe others by one or more blanks. This is followed by a line containing a single integer indicating the number of half-steps to transpose the piece (positive numbers indicating the notes should be transposed up, negative indicating it should be transposed down).

The end of input is signaled by a line consisting of the string "`***`".

Output

For each input set, you should print one line containing the sequence of transposed notes, each represented as in the list of 12 given at the start of this problem. Notes should be printed with a single blank space separating them.

Example

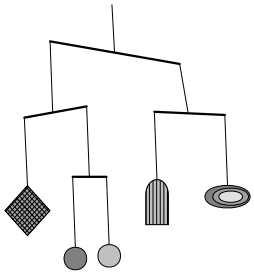
Input:

```
C# E Db G#
1
D E# D A
-1
***
```

Output:

```
D F D A
C# E C# G#
```

Problem B: Mobiles Alabama



Alabama Mobiles Inc. designs and manufactures mobiles, lightweight "kinetic sculptures" consisting of bars hanging from strings. From each end of the bar hangs a string supporting either a small decorative object of some kind or another, smaller mobile. A well-designed mobile must have the weights of all the decorative items balanced, so that each bar in the mobile will naturally tend to hang horizontally. A bar can be balanced even if the weights hanging from either side are not equal. The string must be tied to the bar si that it divides the bar into two lengths L1 and L2, such that

$$L_1 * W_1 = L_2 * W_2$$

where W_1 is the total weight hanging from the L_1 side of the bar and W_2 is the total length hanging from the other side.

Write a program to read a partial design for a new mobile and determine if it can be balanced and if, once balanced, the elements will swing freely without bumping or entangling with one another.

Each mobile will be described as a collection of bars and decorative objects. You will be given the lengths of the bars, the weights of all objects, and information on how the bars and objects are connected to one another. You may assume that the bars are made of a lightweight material so that their weight and the weight of the connecting strings are negligible compared to the weights of the decorative objects.

Every mobile will have at least one bar.

Input

The input consists of a number of mobile design specifications. Each mobile design is given as a parenthesized expression. These expressions are composed of two basic forms:

1. `(` `D` w `)`

   describes a decorative object. w is a floating point number giving the weight of the object.

2. `(` `B` `#` L $\mathtt{m}_1$ $\mathtt{m}_2$ `)`

   describes a bar.

   - `#` is an integer that constitutes a unique identifier for each bar. These integers will be assigned densely so that a mobile with a total of k bars will use the numbers $1 \ldots k$ as identifiers. The order in which these identifiers appear within a total expression is arbitrary.
   - L is a floating point number indicating the length of the bar.
   - $\mathtt{m}_1$ and $\mathtt{m}_2$ are two parenthesized expressions describing the portion of the mobile hanging from each end of this bar.

In the above two forms, wherever a space is shown between the components of each form, the actual input may contain 1 or more spaces and/or line terminators. An exception to this rule is that, to either side of a '(' or ')' in these forms, the input may contain zero or more spaces and/or line terminators.

Every mobile will contain at least one bar.

End of input is indicated by a line containing only: `()`

Output

For each mobile specification, the program should print one line of output for each bar, in order by the object number of the bar. Each line will have the form

`Bar` *N* `must be tied` *L* `from one end.`

where N is the identifying number of the bar and where L is the smaller of the two lengths L_1 and L_2 as described above and is printed to one decimal place precision.

Example

Input:

```
(B 2 4.0
  (D 1.0 )
  (B 1 2.0 (D 1.0 ) (D 2.0 )))
()
```

Output:

```
Bar 1 must be tied 0.7 from one end.
Bar 2 must be tied 1.0 from one end.
```

Problem C: Out of Sight

You fell asleep at your desk (again!) and were awakened by the sound of the newly installed security-cam robots rolling out into the office corridors. You know you'll be in big trouble if any of these robots manages to record your image. Luckily, you are able to quickly call up on your desktop computer the route each robot is programmed to follow. You need to plan your own movements to stay out of a direct line of sight of any of the robots.

The robots move in discrete steps of unit size, moving either to the north, east, west, or south. You may move, also in discrete steps of unit size, in one of these same directions (walls permitting), but you also have the option of remaining in place for a step. You and all robots move simultaneously.

Neither you nor the robots may move through walls, nor can either you or the robots move outside the building.

The robots take a photograph in each of the NSEW directions immediately after moving. To be captured in a photo, you would have to be directly along a NS or EW line from a robot with no walls between you. You are also considered to have been spotted if, at the end of any movement, you are directly on top of a robot.

Input

Input consists of one or more mazes. Each maze begins with a line containing two integers, w and h, denoting the width (west-east) and the height (north-south) of the maze. Neither you nor the robots may leave this area. It is considered to be implicitly enclosed, but may contain other walls. End of input is indicated when either w or h is less than 3.

This is followed by h lines of input, each containing at least w characters. In each of these lines, only the first w characters are significant. Extra characters should be ignored.

The interpretation of the characters in these lines is as follows:

- ' ' denotes an open space
- 'X' denotes a wall.
- 'Y' denotes your current position and will occur exactly once in the maze.
- k, a single digit in the range 0-9 denotes the current position of a robot. No digit will be repeated, and a set of N robots will be denoted by the digits $0 \ldots n-1$.

Each maze will be followed by N lines (where N is the total number of robots). Each line will contain from 0 to 80 characters. All N lines will contain the same number of characters. These chracters will be chosen from among `NSEW` to denote North, South, East, and West, where 'North' points towards the portion of the maze described by the first line of input and 'West' points to the portion of the maze denoted by the first column of each line of the maze input. These characters describe the movement of the robots, one step in the maze at a time.

Output

For each maze, you should print a single line of the form:

`You can hide for` *M* `turns.`

where *M* is the maximum number of steps the robots can take before you are seen. If you can avoid detection during the entire predicted movement of the robots, *M* will be the number of steps given for each robot in the text.

Example

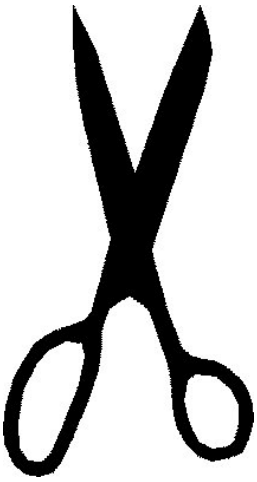
Input:

```
12 7
XXXXXXXXXXXX
X      1   X
X        X X
X        X X
X      XXX X
X    0 XXXYX
XXXXXXXXXXXX
NNEWSWWW
EEEWWWWW
0 0
```

Output:

```
You can hide for 2 turns.
```

Problem D: Witness Redaction Plan



The Justice Dept runs a Witness Protection Plan in which witnesses to crimes are given new identities to protect them from retaliation by the people against whom they testify in court.

Experience has shown that many of the protected witnesses are their own worst enemies - often giving away their new locations and identities in misguided attempts to contact relatives and friends and to assure them that all is well.

The plan is experimenting with a new idea of allowing such communications, in email form only, with the idea that employees of the plan will inspect the communications first and cut out any potentially dangerous sentences, then send the email on from Dept computers so that they cannot be traced back to the witness.

Unfortunately, the budget for the pilot project was cut almost as soon as the project commenced. There is insufficient money to actually hire people to read all the email, so an automated solution is sought instead.

Write a program that, given a list of sensitive words and a message (in plain text form), scans the message for any sentence containing a sensitive word (ignoring differences in upper/lower case). If a sensitive word is found, every character in that sentence (except for line terminators) should be replaced by '@' characters.

For the purposes of this program, a word is a string of consecutive alphanumeric characters bounded in the message by the start or end of the message and/or by any non-alphanumeric character. A sentence is a string of consecutive characters bounded by the start or end of the message, by a paragraph boundary (a line containing zero characters), and/or by one of the punctuation characters: '.', '?', or '!'

Input

Input to the program consists of one or more input sets. Each input set consists of a word list and a message. The end of the input is signaled by a line containing only the left-justified phrase "EndOfInput".

A word list consists of zero or more words, one per line and left-justified. Words may be up to 40 characters in length.The end of the word list is signalled by a line containing only the left-justified phrase "EndOfList".

The word list is immediately followed by a message. A message consists of zero or more lines of text containing up to 80 characters per line. The end of the message is signaled by a line containing only the left-justified phrase "EndOfMsg".

Output

For each input set, print the message, exactly as it appears in the input except for replacement of sentences by '@' characters as described above. At the end of each message, print a line containing only a string of four '=' characters, left-justified.

Example

Input:

```
John
jane
Smith
Jones
Kansas
court
crime
phone
555
EndOfList
Dear Mom,

I just wanted to let you know that I am alive and well. Jane is
well also. I'm glad they were able to relocate us. My
only complaint is that I wish they could have found someplace
more exciting than Kansas for us to live in! If you really need
to contact us, you can do so by telephone. The number is
(757) 555-0478, but don't tell anyone.

Love,
the new John Smith
EndOfMsg
EndOfInput
```

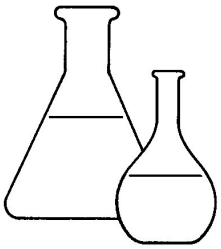
Output:

```
Dear Mom,

I just wanted to let you know that I am alive and well.@@@@@@@@
@@@@@@@@@@ I'm glad they were able to relocate us.@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ If you really need
to contact us, you can do so by telephone.@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@
@@@@@@@@@@@@@@@@@@@
====
```

Problem E: Safety in Alchemy



Frederico de Vinci (one of Leonardo's less talented cousins) is a budding alchemist who spends his days mixing a variety of chemicals hoping to stumble upon the formula to change lead into gold. Near his worktable, he keeps a cheap ceramic jar where he deposits the useless and rather noxious results of his failed experiments.

He has had a few accidents with cracked and shattered disposal jars because of the heat produced by the reacting chemicals he has dropped in there.

Frederico has worked up a table of the amount of the temperature changes produced by various pairs of chemicals he has previously put into his disposal jars. (He believes that simultaneous interactions of 3 or more chemicals are insignificant.)

Write a program to read this information and a list of what he is planning to dispose into the jar, and to print the maximum temperature change that may be produced.

Input

Input consists of one or more problem sets. The final problem set is followed by a line containing the phrase "ENDOFINPUT".

Each problem set begins with a list of up to 64 pairs of chemicals and the heat they produce. This consists of multiple lines, each of the form

chemical1 chemical2 heat

where *chemical1* and *chemical2* are names of chemicals, expressed as an alphabetic string of 1-20 characters. *heat* is an integer in the range $0 \ldots 100$ indicating the number of degrees by which the temperature in the jar will rise when 1 gram of *chemical1* mixes with one gram of *chemical2*. The order in which the two chemicals appear within a single line is not relevant. No pair of chemicals will appear more than once in this list.

This list of pairs is terminated by a line containing "0 0 0". It is followed by a list of up to 12 chemicals that Frederico plans to place in the jar. The list consists of a number of lines of the form

chemical amount

where *chemical* is the name of a chemical (again, expressed as an alphabetic string of 1-20 characters) and *amount* is an integer in the range $0 \ldots 1000$ indicating the quantity (in grams) to be disposed. No chemical will appear more than once in this second list.

This list is terminated by a line containing "0 0".

Output

For each problem set, print a single line of output of the form

`The temperature in the jar will change by at most` *M* `degrees.`

where *M* is an integer indicating the maximum temperature rise.

Example

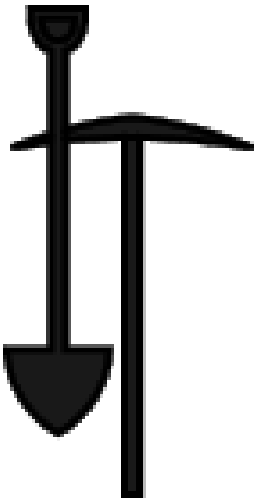
Input:

```
Aqua Beta 2
Beta Calcum 1
Aqua Delta 10
Ente Franca 10
0 0 0
Aqua 50
Calcum 10
Beta 25
Delta 10
Franca 10
0 0
ENDOFINPUT
```

Output:

```
The temperature in the jar will change by at most 150 degrees.
```

Problem F: Potholes



You are competing with another company for a city contract to fill potholes in the city streets.

The city has set up a contest in which you will compete in a rectangular parking lot containing a large number of potholes to see which company can work faster.

You are to stretch a rope from one end of the lot to the other, dividing the lot into two rectangles. Your opponent will then choose which side of the lot they wish to work in, while you work in the other side. Obviously, then, you need to try can divide the lot so that the amount of work to be done (total area of all potholes) on each side is as nearly equal as possible.

For the purposes of this problem, potholes will be modeled as circles. Potholes do not overlap one another or the edges of the parking lot. Your chosen placement of the rope may not cross a pothole. Potholes may, however, be tangent to one another, to the edges of the lot, or to the rope.

Input

Input consists of one or more problem sets. Each problem set consists of

- One line containing the width (x) and height (y) of the parking lot, given as two positive floating point numbers.
- Two or more lines containing descriptions of potholes. Each line consists of three non-negative floating point numbers denoting the (x,y) coordinates of the center of the pothole, followed by the radius of the pothole. The origin of the coordinate system is at a corner of the parking lot.
- The end of an input set is signaled by a line containing 3 zeroes separated by blanks. The end of all input is signaled by a subsequent line containing two zeros separated by blanks.

Write a program that, given a description of the parking lot and of the potholes, determines where to place to rope so that the total area of the potholes on each side of the rope is as nearly equal as possible. Potholes will be modeled as perfect circles. The rope may not pass through any pothole, though it may be tangent to potholes.

Output

For each problem set, print a line

```
$x_1$ $y_1$ $x_2$ $y_2$
```

where (x_1, y_1) and (x_2, y_2) are the points where the rope crosses the boundary of the parking lot. Order your output so that $x_1 \leq x_2$ and $y_1 \leq y_2$. Point coordinates should be printed to one decimal place accuracy.

These points should be chosen so as to most nearly divide the set of potholes into two equal sets by total area.

If there is more than one possible placement of the rope that achieves the most nearly equal divisions of pothole areas, choose the placement that most nearly divides the parking lot into equal areas. If this does not resolve the tie, favor the placement of the rope that intersects the x axis of the lot. Any remaining ties should be broken in favor of the smaller x value.

Example

Input:

```
16.0 12.0
1.0 1.0 0.8
8.0 6.0 2.0
3.0 5.0 1.0
3.0 9.0 1.0
0 0 0
0 0
```

Output:

```
6.0 0.0 6.0 12.0
```

Problem G: Bulls and Cows

Bulls and Cows is a game played between two players, one called the *codemaker* and the other the *codebreaker*. The codemaker forms a secret "code", a decimal integer of K digits, all of them different. The codebreaker then attempts to guess the code by naming numbers of K digits. For each guess, the codemaker responds with the number of digits of correct value in the correct position ("Bulls") and the number of correct digits in an incorrect position ("Cows").

For example, if the secret code were 1230 and the codebreaker guessed 1205, the response would be "2 Bulls and 1 Cow".

Write a program to read a series of guesses and responses for a game and to print the number of possible solutions and, from among all possible solutions, the numerically smallest one.

Input

Input consists of one or more game descriptions.

Each game description begins with a line with a single integer K in the range $1 \ldots 7$. A non-positive value for K signals the end of input.

Each positive K is followed by some number of lines, each line containing one guess and response. Each line will contain K+2 single-digit numbers, separated by one or more blanks. The first K digits represent the guess. The next number represents the number of Bulls. The final digit is the number of Cows.

The end of the sequence of guess-and-response lines is signaled by a line of K integers, the first of which is -1.

Output

For each game description in the input, print a line of the form

`N is one of M possible solutions.`

where the *N* is a sequence of K digits, printed with no intervening spaces, and representing the numerically smallest possible solution, and where *M* is an integer denoting the number of possible solutions.

Example

Input:

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

Output:

```
012345 is one of 5 possible solutions.
```

Problem H: The Turn of the Shrew

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.

In recent years he has begun to suspect that the shrew population has been undergoing a high rate of genetic mutation (possibly due to that strange glowing rock near their communal burrow). When the shrews emerged from their burrow at the end of the past winter, he noted quite a few new youngsters whose physical characteristics did not match up with any possible pair of prospective parents.

Write a program to determine the smallest number of mutations for each juvenile shrew that would account for its possible parentage.

Input

Input consists of one or more input sets.

Each input set consists of a number of lines, each describing a single animal. Each line begins with a single character, either 'M', 'F', or 'C'. These denote a male adult, a female adult, and a child, respectively. There will be at least one of each type of record in an input set. This initial character is followed by a blank, then by 1-40 consecutive 0 or 1 characters describing a genetic code for that animal. All lines in a given input set will have the same number of characters in their genetic codes.

A 1 indicates that the animal possesses a particular physical characteristic associated with a dominant gene, a 0 indicates that it does not. The input set is terminated by a line beginning with the character 'X'. The end of all input sets is indicated by a second, consecutive line beginning with 'X'.

In the absence of mutation, a child can have a 1 in a gene position only if at least one parent has a 1 there, and can have a 0 in that position only if both parents have a 0 in the corresponding position. Assume that a mutation affects only a single genetic marker.

Output

For each child, print a line of the form

`Child` *K* `has a minimum of` *N* `mutations.`

where *K* is the number (beginning with 1) indicating the order in which the child appeared in the input (*K* ascending with each output line) an where *N* is the minimum number of mutations that could account for the birth of this child from some pair of adults.

Example

Input:

```
M 0011
F 0000
M 1001
C 1011
F 0110
C 1111
X
X
```

Output:

```
Child 1 has a minimum of 1 mutations.
Child 2 has a minimum of 0 mutations.
```