2008 Mid-Atlantic Regional Programming Contest

Welcome to the 2008 Programming Contest. Before you start the contest, please be aware of the following notes:

The Contest

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	Close Enough Computations	Yellow
B	I-Soar	Green
C	Trie, Trie Again	Silver
D	Lawrence of Arabia	Black
E	Series/Parallel Resistor Circuits	Orange
F	Combination Lock	Purple
G	Stems Sell	Pink
H	Signal Strength	Red

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.

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

Your Programs

8. 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 (cerr in C++ or System.err in Java). 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
   ```

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. 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.

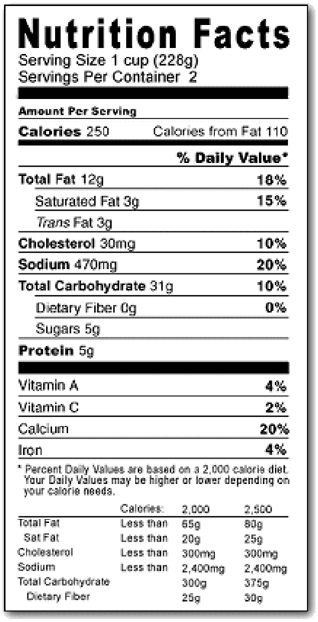
12. If a problem specifies that an input is a floating point number, decimal points may be omitted for numbers with no non-zero decimal portion. Scientific notation will not be used in input sets unless a problem explicitly allows it.

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.

Good luck, and HAVE FUN!!!

Problem A: Close Enough Computations



Nutrition Facts
Serving Size 1 cup (228g)
Servings Per Container 2

Amount Per Serving

Calories 250 Calories from Fat 110

	% Daily Value*
Total Fat 12g	**18%**
Saturated Fat 3g	**15%**
Trans Fat 3g	
Cholesterol 30mg	**10%**
Sodium 470mg	**20%**
Total Carbohydrate 31g	**10%**
Dietary Fiber 0g	**0%**
Sugars 5g	
Protein 5g	
Vitamin A	**4%**
Vitamin C	**2%**
Calcium	**20%**
Iron	**4%**

* Percent Daily Values are based on a 2,000 calorie diet. Your Daily Values may be higher or lower depending on your calorie needs.

	Calories:	2,000	2,500
Total Fat	Less than	65g	80g
Sat Fat	Less than	20g	25g
Cholesterol	Less than	300mg	300mg
Sodium	Less than	2,400mg	2,400mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g

The nutritional food label has become ubiquitous. A sample label is shown to the right. On the label the number of calories and the number of grams of fat, carbohydrate, and protein are given as integers.

But carefully reading the label may cause the consumer to notice some inconsistencies. A gram of fat has 9 calories, a gram of carbohydrate has 4 calories, and a gram of protein has 4 calories. Consider the label to the right. A simple computation of the number of calories would indicate that the food should contain $12 * 9 + 31 * 4 + 5 * 4$ or 252 calories, but the label indicates it has 250 calories.

While sometimes the difference in calories is due to other circumstances (such as the presence of alcohol or soluble fiber), this problem will consider only the possibility of round-off error. This food actually has 12.1 grams of fat (yielding 108.9 calories), 30.6 grams of carbohydrate (122.4 calories), 4.7 grams of protein (18.8 calories), so it does in fact have 250 calories (actually 250.1 calories).

Write a program that will determine if values for a nutritional label are consistent, that is, if there is a way the true values for the grams of nutrients can be rounded to the shown values and yield the number of calories shown.

You should assume that standard rounding rules apply; that is any value less than 0.5 rounds down and those 0.5 or over round up.

Input

The input will contain one or more sets of data about potential labels. Each data set will consist of 4 non-negative integers, separated by one or more blanks, on a single line. The integers represent the number of calories, the number of grams of fat, the number of grams of carbohydrates, and the number of grams of protein, in that order. The number of calories will not exceed 10000, and the number of grams of any component will not exceed 1000.

End of input is indicated by a line containing 4 zeroes.

Output

For each data set, print "yes" or "no" on its own line, indicating whether the given rounded values of the three nutrients can yield the given number of calories.

Example

Input:

Given the input

```
250 12 31 5
250 13 31 5
```

```
122 10 10 0
0 0 0 0
```

the output would be

Output:

```
yes
no
no
```

Problem B: I-Soar

The town meeting was not going well. ”It’s noisy”, some town residents complained. ”It’s ugly”, others stated. ”It’s an eyesore”, many agreed. ”It’s here,” said the mayor, ”and it’s not going to go away.”

The cause of all this furor was the new stretch of Interstate Highway that had been just opened. Straight as an arrow, it ran along the entire northern edge of the town.

”Look,” said the mayor, ”we can reduce the noise and improve the view by planting trees and tall hedges along the road, but we don’t have an unlimited budget. Luckily, much of the highway is already hidden by some of the buildings in our commercial district on the north. We’ll see what we can do by planting in the visible gaps between the buildings.”

Write a program to compute the total linear length of planting that will be required to block the view of the Interstate by an observer looking straight north (orthogonal to the highway) from the southern side of the commercial district.

Input

Input consists of multiple data sets. The first line in each data set contains the length of the town border adjacent to the highway, expressed as a floating point number (called L, below). A non-positive value for this number signals the end of input. This is followed by zero or more lines containing the positions of buildings within the commercial district. Each such line gives a pair of x positions (floating point numbers) representing the portion of the interstate whose view is occluded by the building. These numbers are expressed in the same units of measurement as the length of the border, such that 0 denotes the western end of the border and L the eastern end. The end of a data set is signaled by any pair $x1, x2$ for which $x1 > x2$.

Output

For each data set, print one line of the form

```
The total planting length is ##
```

where ## is a floating point number, printed to one decimal place precision, denoting the total length of the Interstate visible between the buildings.

Example

Input:

Given the input

```
100.0
20.0 30.0
 40.0 50.0
1.0 0.0
```

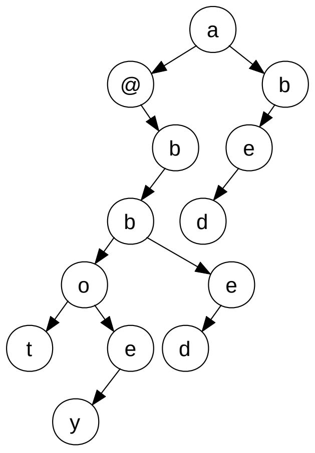
```
100.0
20.0 30.0
22.0 28.0
1.0 0.0
-1.0
```

the output would be

Output:

```
The total planting length is 80.0
The total planting length is 90.0
```

Problem C: Trie, Trie Again



Spark Plug Searching, Ltd., is looking to enter the market of web-based applications with a new word processor. The team implementing the spell checker have proposed storing the dictionary in a binary-encoded trie, a binary tree structure in which words are located by moving down the tree, one letter at a time, but discarding any letter from which one takes a right branch.

The end of a word is indicated whenever a letter has no left child or by an @ character. This trie (in the diagram) contains the words a, abbot, abbey, abed, and bed.

Some team members have objected that the resulting data structure will be too large for practical use. Others have argued that the size can be dramatically reduced by taking advantage of the fact that many words end in common suffixes. The trie can be scanned for common subtrees and the parents of identical subtrees could be altered so that they all point to a single instance of that subtree. Your job is to demonstrate this process. Write a program to find the shared subtree that would produce the maximum reduction in the number of nodes if all instances of that subtree were replaced by a single shared instance.

Input

Input consists of one or more lines, each line describing one tree. Each tree is presented, left justified, in preorder form using at least one and up to 200 characters. The nodes of the tree are denoted by a single lowercase letter or @. The character # indicates that a node does not have a child in the indicated position. Note that with these conventions, the tree described by the preorder traversal will be unique.

End of input is indicated by a line consisting of the word "END".

Output

For each input tree, print the non-empty subtree (in the same format) that yields the greatest savings if all instances are replaced by a single shared instance. Then, on the same line, print a space and then the number of nodes that would be saved by this replacement. Print no blank lines between outputs.

If there is a tie among different subtrees that could yield maximal savings, select the smallest tree from among those tied. If there remains more than one candidate, select the one occurring first in a preorder traversal of the original tree.

Example

Input:

Given the input

```
a@#bbot#ey##ed###bed#######
END
```

the output would be

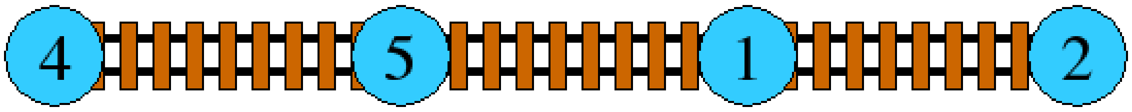
Output:

```
ed### 2
```

Problem D: Lawrence of Arabia

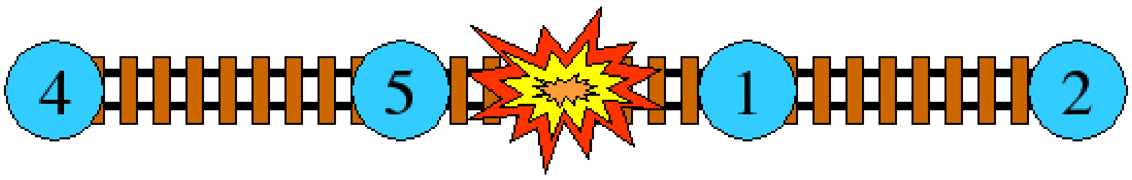
T. E. Lawrence was a controversial figure during World War I. He was a British officer who served in the Arabian theater and led a group of Arab nationals in guerilla strikes against the Ottoman Empire. His primary targets were the railroads. A highly fictionalized version of his exploits was presented in the blockbuster movie, "Lawrence of Arabia".

You are to write a program to help Lawrence figure out how to best use his limited resources. You have some information from British Intelligence. First, the rail line is a completely linear—there are no branches, no spurs. Next, British Intelligence has assigned a Strategic Importance to each depot—an integer from 1 to 5. A depot is of no use on its own; it only has value if it is connected to other depots. The Strategic Value of the entire railroad is calculated by adding up the products of the Strategic Values for every pair of depots that are connected, directly or indirectly, by the rail line. Consider this railroad:

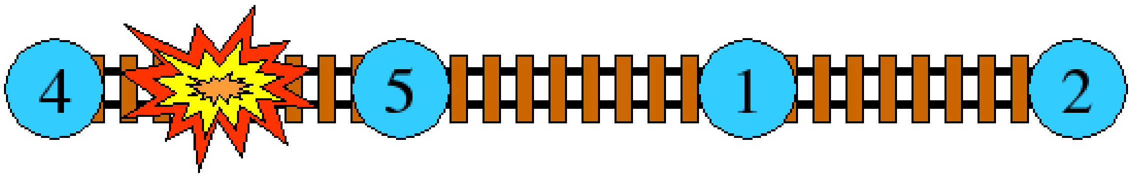


Its Strategic Value is $4 * 5 + 4 * 1 + 4 * 2 + 5 * 1 + 5 * 2 + 1 * 2 = 49$.

Now, suppose that Lawrence only has enough resources for one attack. He cannot attack the depots themselves—they are too well defended. He must attack the rail line between depots, in the middle of the desert. Consider what would happen if Lawrence attacked this rail line right in the middle:



The Strategic Value of the remaining railroad is $4 * 5 + 1 * 2 = 22$. But, suppose Lawrence attacks between the 4 and 5 depots:



The Strategic Value of the remaining railroad is $5 * 1 + 5 * 2 + 1 * 2 = 17$. This is Lawrence's best option.

Given a description of a railroad and the number of attacks that Lawrence can perform, figure out the smallest Strategic Value that he can achieve for that railroad.

Input

There will be several data sets. Each data set will begin with a line with two integers, n and m. n is the number of depots on the railroad ($1 \leq n \leq 1000$), and m is the number of attacks Lawrence has resources for ($0 \leq m < n$). On the next line will be n integers, each from 1 to 5, indicating the Strategic Value of each depot in order. End of input will be marked by a line with $n = 0$ and $m = 0$, which should not be processed.

Output

For each data set, output a single integer, indicating the smallest Strategic Value for the railroad that Lawrence can achieve with his attacks. Output each integer in its own line.

Example

Input:

Given the input

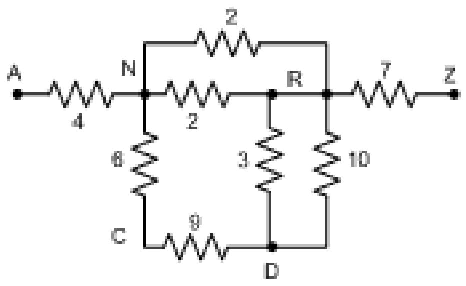
```
4 1
4 5 1 2
4 2
4 5 1 2
0 0
```

the output would be

Output:

```
17
2
```

Problem E: Series / Parallel Resistor Circuits



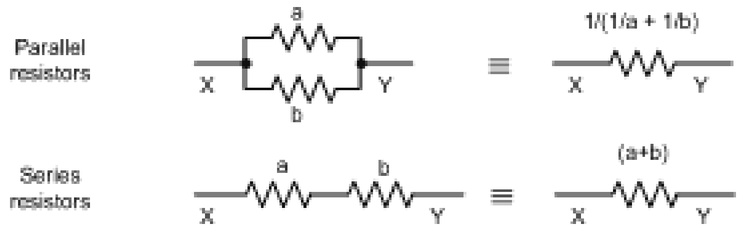
A series/parallel resistor circuit is shown here.

The resistance value is given next to each resistor. Connection points (wires connecting two or more resistors together, are denoted by an uppercase letter. A and Z are reserved for the names of the connection points which are the endpoints of the circuit. Our goal is to calculate the equivalent resistance of the circuit (i.e., the equivalent resistance between A and Z).

Within the circuit, a resistor can be specified by a triple consisting of the connection points at either endpoint, and the resistance. The resistor labelled "9" could be specified as either (C, D, 9) or (D, C, 9). A circuit specification is the set of all resistor specifications.

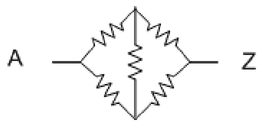
A pair of resistors is in series if one of either of their endpoints have a common connection point that is not use by any other resistor (e.g., the resistors labelled "6" and "9" are both connected to C, which is not connected to anything else). Two series resistors can be replaced by an equivalent single resistor whose resistance is the sum of the replaced resistors (15, in the previous example).

A pair of resistors is in parallel if both their endpoints have common connection points (e.g., the resistors labelled "3" and "10" above are both connected to R and D). Two parallel resistors can be replaced by an equivalent single resistor whose resistance is the inverse of the sum of the inverses of the two resistors ($(1/3 + 1/10)^{-1} = 2.307692$, in the previous example).



The equivalent resistance of a well-formed series-parallel resistor circuit can be determined by successively replacing a series or parallel resistor pair by the single equivalent resistor, until only one is left.

Not all circuits can be decomposed into series and parallel components. The Wheatstone Bridge, shown here, is a classic example of a circuit that is not considered a well-formed series-parallel resistor circuit.



Input

There will be multiple circuit specifications. The first input line for each circuit specification is an integer N ($N < 1000$), the number of resistors in the circuit. This is followed by N lines, each being a resistor specification in the form: XYr, where X and Y are uppercase characters, and r is a positive integer resistance ($r < 100$). The equivalent resistance is guaranteed never to be greater than 100.

A circuit with $N = 0$ indicates the last circuit, and should not be processed.

Output

For each circuit, print a line of the form

```
Circuit k: r
```

where *k* is the circuit number in the input, starting at 1. If the circuit is a well-formed series-parallel circuit, *r* should be the equivalent resistance from A to Z. If the circuit is not well formed, *r* should be the number -1.000. In either case, *r* should be displayed with 3 decimal places precision.

Example

Input:

Given the input

```
8
N R 2
D R 3
R N 2
R D 10
Z R 7
C D 9
N C 6
A N 4
2
A Z 3
Z A 10
2
P A 6
P Z 9
5
A B 1
B Z 4
A C 8
C Z 19
B C 12
0
```

the output would be

Output:

```
Circuit 1: 11.945
Circuit 2: 2.308
Circuit 3: 15.000
Circuit 4: -1.000
```

Problem F: Combination Lock

A combination lock consists of a circular dial, which can be turned (clockwise or counterclockwise) and is embedded into the ”fixed” part of the lock. The dial has N evenly spaced ”ticks”. The ticks are numbered from 0 to $N - 1$, increasing in the clockwise direction. The fixed part of the lock has a ”mark” which always ”points to” a particular tick on the dial. Of course, the mark points to different ticks as the dial is turned.

The lock comes with three code numbers $T1$, $T2$, $T3$. These are non-negative integers and each of them is less than N. No two of the three are the same.

The lock is opened in three stages of operations:

1. Turn the dial clockwise exactly two full revolutions, and continue to turn it clockwise until the mark points to tick $T1$.
2. Turn the dial one full revolution counterclockwise and continue to turn it counterclockwise until the mark points to tick $T2$.
3. Turn the dial clockwise until the mark points to tick $T3$. The lock should now open.

You must find the maximum possible number of ticks the dial must be turned in order to open the lock. The number of ticks turned is defined to be the sum of the ticks turned in the three stages outlined above, and is always positive regardless of direction.

Input

The input file consists of a number of test cases, one test case per line. Each line of the input file contains four integers: N, $T1$, $T2$, $T3$, in this order, separated by blank spaces. The integer N is a multiple of 5, $25 \leq N \leq 100$. The numbers $T1$, $T2$ and $T3$ satisfy the constraints stated under the description above. The input will be terminated by a line with $N = T1 = T2 = T3 = 0$.

Output

For each test case, print the maximum possible number of ticks the dial must be turned in order to open the lock. Print each on its own line.

Example

Input:

Given the input

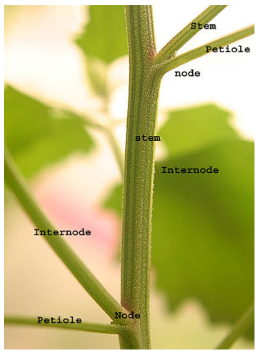
```
80 20 40 50
80 10 79 12
0 0 0 0
```

the output would be

Output:

```
409
455
```

Problem G: Stems Sell



The designers at Spark Plug Searching, Ltd., are searching for additional ways to reduce the size of the dictionary used for their new spell checker. Their latest idea is that maybe they can get by storing only the basic "stem" form of a word without including the plural forms, past tenses, and other grammatical variants of the stem word.

When a word is extracted from a document and we want to check it for correct spelling, we can apply a series of rules to rewrite the word to its likely stem form. Then we check the dictionary to see if that rewritten word matches any entry in the dictionary.

Natural languages are pretty messy and it may take quite a bit of experimenting to get an appropriate set. So the team has proposed a simple language for describing rewrite rules. A rule has the form

```
pattern => replacement
```

A *pattern* is a sequence of one or more characters which are interpreted as follows:

- Lower-case alphabetic characters in the pattern match a single occurrence of that same character in either upper or lower case. For example, a pattern "ab" can match and "ab", "aB", "Ab" or "AB" in a word.
- An asterisk (*) matches one or more alphabetic characters. There is at most one asterisk in a pattern, and if one is present it will be the first character in the pattern.
- An upper-case 'V' matches any lower-case vowel ('a', 'e', 'i', 'o', or 'u').
- An upper-case 'C' matches any lower-case consonant (any alphabetic character other than 'a', 'e', 'i', 'o', or 'u').
- A numeric digit 1-9 matches the same string as was matched by the character in that position of the pattern, counting the first character of the pattern as character #1. A digit k can occur in the pattern only after position k.

For example, the pattern `*C2ies` would match any word of at least 6 characters ending in two copies of the same consonant followed by "ies"".

The *replacement* part of a rule is built from a limited set of the same characters. Specifically, the replacement can contain only lower-case alphabetic characters (which are kept "as is") and numeric digits, which are replaced by the same string they would have matched in the pattern. For example, the rule

```
*C2ies => 122y
```

applied to the word "berries" would match the "*" against "be", the "C" and "2" against "r", and the "ies" against "ies". In the replacement, the "1" refers to the "*" which matched "be", the "2"s each match "r", and the "y" is left as is, so the word "berries" is rewritten to "berry".

Write a program to read a set of rewrite rules and to apply them to words in a paragraph of text. For each word (maximal consecutive string of alphabetic characters) in the paragraph, apply the rules, one at a time, in the order they are given, until a matching pattern is found or until all rules are exhausted. If a matching pattern is found, apply that rule to rewrite the word. If not, leave the word unchanged.

Input

Input consists of multiple data sets. Each data set begins with a set of one or more rules, one per line in the format described above, followed by an empty line. This is followed by one or more lines of text, followed by an empty line or by a line containing only the left-justified string ***, which indicates end of the input

Output

For each data set, print the lines of text with each word rewritten according to the given rule set, and all non-word characters left unchanged.

At the end of the rewritten text for each data set, print a single line containing the left-justified string ***.

Example

Input:

Given the input

```
*ed => 1

The quick brown foxes jumped over
the lazy yellow dogs.

*ed => 1
*ies => 1y
*s => 1

The quick brown foxes jumped over
  the lazy yellow dogs.
***
```

the output would be

Output:

```
The quick brown foxes jump over
the lazy yellow dogs.
***
The quick brown foxe jump over
  the lazy yellow dog.
***
```

Problem H: Signal Strength

Net Profits Incorporated has announced a new generation of network switching devices aimed at supporting high-speed, instantly reconfigurable networks over long distances.

Their networks are based on a new kind of switching device that monitors several input lines, locking on to the strongest signal that it receives and passing that signal on to all output lines connected to its output ports.

The connecting lines are long enough that signal loss is a major concern. The switches themselves may cause additional signal loss. To counter both sources of loss, some switches are equipped with amplifiers. (To prevent destructive feedback, switches with amplifiers will not be connected in cycles such that their output can reach their own inputs, even indirectly.)

Develop a program to predict the effective signal strength that can be expected when a signal initiated at one point in a network is received at another point. You will be provided with a description of a network consisting of N switches ($1 \leq N \leq 1000$). For each switch, you will given a multiplier indicating how much weaker or stronger the output of the switch will be than the strength of the strongest input coming in to that switch. Switches without amplifiers will multiply the signal strength by a factor of no smaller than 0.1. Switches with amplifiers may multiply the signal strength by factors as high as 5.0.

You will also be provided with a description of the connecting lines within the network, including a multiplier indicating how much weaker a signal is at the end of the line than when it entered. These multipliers will be no smaller than 0.1 and no larger than 1.0.

Input

Input consists of one or more input sets.

Each input set describes one network. It begins with a line containing one integer, N, the number of switches in the network. These switches are identified by number starting at 0. The end of input is signaled by a non-positive value for N.

This is followed by N lines, each describing one switch and the lines attached to the output of that switch. This description begins with a floating point number giving the strength multiplier for that switch. This is followed by an integer k indicating how many lines are connected to the output of the switch. After that are k pairs of numbers, each describing one output line. The first number in each pair gives the number of the switch to which this output line connects (i.e., the switch receiving this line as an input). The second number is a floating point number giving the multiplier describing the signal loss on that line.

Output

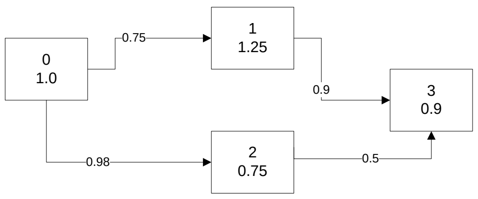
For each network, print a line of the form

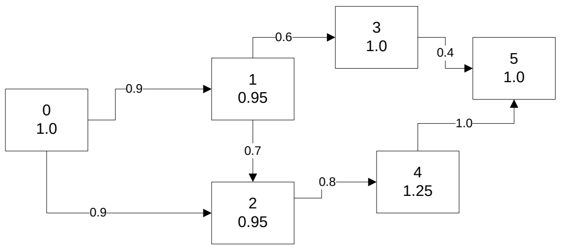
```
Network M: X
```

where *M* is the input set number (starting at 1) and where *X* is the signal strength expected at the output for switch $N-1$ if an input signal of strength 1.0 is presented at (all) inputs of switch 0. If that input never appears at the output of switch $N-1$, then *X* should be zero. *X* should be printed to two digits precision.

Example

The networks shown here are described by the following input and output.





Input:

Given the input

```
4
1.0 2 1 0.75 2 0.98
1.25 1 3 0.9
0.75 1 3 0.5
0.9 0
6
1.0  2  1 0.9  2 0.9
0.95 2  2 0.7  3 0.6
0.95 1  4 0.8
1.0  1  4 0.4
1.25 1  5 1.0
1.1  0
0
```

the output would be

Output:

```
Network 1: 0.76
Network 2: 0.94
```