Preamble

Please note the following very important details:

1) Read all input from the keyboard, i.e. use `stdin`, `System.in`, `cin` or equivalent. Input will be redirected from a file to form the input to your submission.

2) Write all output to the screen, i.e. use `stdout`, `System.out`, `cout` or equivalent. Do not write to `stderr`. Do NOT use, or even include, any module that allows direct manipulation of the screen, such as `conio`, `Crt` or anything similar. Output from your program is redirected to a file for later checking. Use of direct I/O means that such output is not redirected and hence cannot be checked.

   In the C programming language, the use of the `gets()` function from `stdio` library is deprecated. Use `fgets` with the file stream `stdin` instead.

3) Unless otherwise stated, all integers will fit into a standard 32-bit computer word. If more than one integer appears on a line, they will be separated by white space, i.e. spaces or tabs.

4) Unless otherwise stated a word is a continuous sequence of lower case letters without any punctuation or other characters and, in particular, without intervening white space. As with numbers, successive words will be separated by white space.

Problem A Pig Latin 10 Points

Pig Latin is an old scheme used, typically by children, to render speech unintelligible to outsiders. The fact that this is seldom necessary seems to escape every generation.

The basic rule is: move the initial segment of a word, up to but not including the first vowel, to the end of the word, and append ‘ay’, thus ‘frog’ becomes ‘ogfray’ and ‘apple’ becomes ‘appleay’. If the word starts with a vowel, or there is no vowel present, append ‘ay’ to the unchanged word.

Input will be a list of words, one per line, terminated by a line containing a single ‘#’. Each word will consist of no more than 20 lower case letters with no embedded white space. Note that the vowels in this context are assumed to be ‘a’, ‘e’, ‘i’, ‘o’, and ‘u’.

Output will be a list of words, one per line. Each word will be the Pig Latin equivalent of the corresponding word in the input.

Sample Input

```
frog
apple
pear
#
```

Sample Output

```
ogfray
appleay
earpay
```

Problem B Political Correctness 10 Points

There are many terms that people find offensive and nowadays it is easy to automate the process of eliminating these words, regardless of where they may occur. These offensive words are usually called four-letter words, chiefly because they are, but for this program we will look at any successive four letters, regardless of whether they are a word on their own or part of a larger word.

Of course, the exact terms that are deemed to be offensive depends on the listener — I am sure there are many of you who find the word 'work' at least disturbing, if not actually offensive. Thus the list of terms will be specified. Further, because these terms are so offensive, they cannot **ever** be specified explicitly, so they will in fact be referred to their first and last letters only.

For example, if the list of offensive words included 'st', 'fk', 'dn', and 'ct', then the sentence:
`‘I cantered down to the shuttered shop to buy a fork.’` becomes
`‘I c**tered d**n to the s**ttered shop to buy a f**k.’`

Input will consist of a 'dictionary', a list of no more than 20 words specified as pairs of lower case letters terminated by a line containing two # characters. This will be followed by a paragraph to be sanitised. Each line of the paragraph will contain no more than 60 characters. No word will straddle a line break. The paragraph will be terminated by a # on a line by itself.

Output will be the sanitised version of the given paragraph. Replace all sequences of four letters (no white space, punctuation marks or other characters) which are bounded by one of the pairs given in the dictionary, even if the cases differ, by ** in the central positions. All other characters, including formatting characters such as tabs and new line characters are to be left untouched. Words will be processed sequentially and overlapping sequences need not be considered.

Sample Input

```
st
fk
dn
ct
##
I cantered down to the Shuttered shop’s to buy a forK.
#
```

Sample Output

```
I c**tered d**n to the S**ttered shop’s to buy a f**K.
```

Problem C Check Digits 10 Points

Many items, from books to groceries, from bank accounts to credit cards and practically everything in between are identified primarily by a number, often involving many digits. As you can imagine, it is very easy to make a mistake when transcribing such numbers, thus most such numbers incorporate a mechanism to detect, and possibly correct, errors.

The simplest and easiest scheme calculates a single check digit by multiplying the rightmost digit by 2, the next digit by 3 and so on and then forming the sum. This sum is then divided by 11 and the remainder is subtracted from 11. If this is a number in the range 1 to 9 then it is appended to the right end of the number. If it is equal to 11, the digit 0 is appended as the check digit, and if it is equal to 10, then the original number is rejected.

To check whether a complete number is correct, multiply successive digits, from the right, by 1, 2, 3, etc. and form the sum. If this sum is divisible by 11 then the number is good otherwise it is bad.

As an example, consider the number 2763. To generate the check digit, multiply 3 by 2 (6), multiply 6 by 3 (18), and add (24), multiply 7 by 4 (28) and add (52) and multiply 2 by 5 (10) and add (62). Divide 62 by 11 to give a remainder of 7. Subtract 7 from 11 to give the check digit 4. Thus the full number would be 27634. I will leave you to check that this works the other way and that changing any digit (or even reversing two digits) will cause the number to be wrong.

Write a program that will read in a series of numbers (up to 15 digits long) and then generate check digits for them

Input will be a series of numbers, one per line. Each number will contain at least one and no more than 15 decimal digits without embedded whitespace. The file will be terminated by a line containing a #

Output will be a series of lines, one for each number in the input, except for the terminating zero. Each line will consist of the original number followed by the characters ‘` -> `’ followed by either the check digit or the word ‘`Rejected`’.

Sample Input

```
2763
0
```

Sample Output

```
2763 -> 4
```

<<We will obviously need to generate more of this. I will do it once I have a working program :-) >>

Problem D Word Ladder 10 Points

There is a class of word puzzles where you are given two words, such as BEAK and MAKE, and have to get from one to another by changing one letter at a time. Solving such puzzles requires a good vocabulary and some lateral thinking, but checking the solution once you have one is merely tedious and suitable for a computer to do.

A solution is correct if, for each pair of adjacent words in the ladder, the following apply:

- they are the same length
- there is exactly one letter changed.

Write a program that will check a proposed solution. Note that even correct solutions are not guaranteed to be minimal. Input will be a series of lines each containing a single word terminated by a line containing a single #. A word is a sequence of between three and twenty uppercase letters. The file will be terminated by an empty ladder, i.e. another #.

For each word ladder in the input, output the word '`Correct`' or '`Incorrect`' as appropriate.

Sample input

```
BARK
BARE
#
BEAK
BRAK
BRAD
BEAD
#
BEAK
BEAD
BEND
LEND
LAND
LANE
LAKE
#
MAKE
BAKE
BONK
BONE
BANE
#
#
```

Sample output

```
Correct
Correct
Correct
Incorrect
```

Problem F TAXI ROUTES 30 Points

In the town of Gridville the road network is a perfect rectangular grid. Since the founders of Gridville were computer scientists, this grid is numbered starting from 0 in both the EW and NS directions. The roads running EW are called streets, and those running NS are called avenues. A taxi company has its depot in the SW corner of the grid (i.e. at the intersection of 0th St and 0th Ave.) The problem is to determine how many routes are available to the NE corner of the city driving only in an eastward or northward direction (that is, no backtracking is allowed). The situation is complicated by the fact that certain intersections are under construction and are therefore impassable. Note that we will guarantee that the number of routes will never exceed 2147483647 (2^31-1).

Input will consists of a sequence of 'maps'. Each map begins with a line consisting of a pair of integers in the range from 1 through 30 inclusive giving the number of streets and avenues respectively. This is followed by a sequence of lines also containing pairs of integers which denote the impassable intersections (the first element of a pair is the street number, the second the avenue number). Note that neither the home nor the destination intersections will appear on this list. Input for a single map is terminated by the pair 0 0. Input as a whole is terminated by another line containing pair 0 0.

For each map in the input, output a single line in the following form:

```
Map <mapId>: <num>
```

Here `<mapId>` is the identification number of the map (an integer, beginning from 1), and `<num>` is the number of routes available. Note that `<num>` will never exceed 2147483647 (2^31-1).

Sample Input

```
3 3
0 0
5 5
1 1
2 2
3 3
0 0
4 4
1 0
0 1
0 0
0 0
```

Sample Output

```
Map 1: 6
Map 2: 10
Map 3: 0
```

Problem G URNS 30 Points

Initially you are given five urns, each containing balls of a single colour, those colours being red, orange, yellow, green, and blue. Balls are then transferred from urn to urn. The problem is to report the contents of each urn at the end of the process.

The urns are well-mixed before each transfer, so well-mixed in fact that, as nearly as possible, the number of balls transferred of each colour will match their relative proportions in the source urn before the transfer.

For example, if an urn contains 60 red balls, and 40 green balls, and 10 balls are transferred then exactly 6 will be red, and 4 will be green. If 12 balls are transferred then:

(60/100)*12 = 7 + 20/100 should be red, and
(40/100)*12 = 4 + 80/100 should be green.

In this case 7 red and 5 green balls will be moved since the discrepancy that this produces from the ideal arrangement is:
| 7 - (7 + 20/100) | + | 5 - (4 + 80/100) | = 20/100 + 20/100 = 40/100
which is smaller than the discrepancy produced by any other move.

In some cases there might be two moves of equal discrepancy. For example if an urn contains 50 each of red, green, and blue balls, and two are drawn then choosing two balls of any two different colours always gives the same discrepancy. To break such ties, we write the choices as sequences of the form (r, o, y, g, b) and choose the smallest one in dictionary ordering. In this case we must choose among (1, 0, 0, 1, 0), (1, 0, 0, 0, 1), and (0, 0, 0, 1, 1), and the choice
we make is the last one.

If an attempt is made to move more balls than are present in an urn, then that simply results in moving all the balls from that urn.

Input will consist of a number of trials. Each trial begins with the name of the trial on a single line. This is followed by a line containing five non-negative integers in the range from 0 through 99999 (inclusive) giving the initial contents of each of the five urns. These lines are followed by a series of lines each consisting of three integers. The first of these is the number of balls to be moved, the second the number (1 through 5) of the source urn, and the third the number of the target urn. Each trial is terminated by a line containing 3 zeroes (0 0 0). The file will be terminated by a line containing only a single #.

For each trial the output consists of the name of the trial followed by the results for that trial. This consists of a heading line consisting of the word '`URN`', then eight spaces, and then the characters '`R`', '`O`', '`Y`', '`G`', '`B`', each separated from the next by six spaces. The following five lines give the final contents of the five urns. Each line should begin with an urn number (1 through 5) in that order, followed by four spaces. Then the contents of that urn are printed as a sequence of five integers, each right justified in a field of width seven. Separate output for each trial by a blank line.

Sample Input

```
No Blue
100 20 50 30 5
50 1 2
20 1 3
17 4 2
31    3 1
0 0 0
All Blue
1 1 1 1 99999
2 1 2
0 0 0
Well Mixed
1 1 1 1 1
1 1 2
2 2 3
3 3 4
4 4 5
0 0 0
#
```

Sample Output

```
No Blue
URN         R      O      Y      G      B
1          39      0     22      0      0
2          50     20      0     17      0
3          11      0     28      0      0
4           0      0      0     13      0
5           0      0      0      0      5

All Blue
URN         R      O      Y      G      B
1           0      0      0      0      0
2           1      1      0      0      0
3           0      0      1      0      0
4           0      0      0      1      0
5           0      0      0      0  99999

Well Mixed
URN         R      O      Y      G      B
1           0      0      0      0      0
2           0      0      0      0      0
3           0      0      0      0      0
4           0      0      0      0      0
5           1      1      1      1      1
```

Problem H Crossword Puzzle Clue Numbering 30 Points

Professor Logophile is the local crossword puzzle setter. He has a very idiosyncratic way of working — he writes the words into a blank grid and then fills in the unused squares. That is the easy bit, but he has trouble doing the next bit — numbering the puzzle and preparing the the clues.

This is where you come in. Given a completed crossword, print out a form on which the clues can be written (showing word lengths). Note that he always prepares a true crossword, i.e. there is always at least one word in each direction and there is always at least one shared letter. All words contain three or more letters.

Input will be a series of crosswords. The first line of each crossword will be a pair of integers (*r* and *c*, $3 \leq$ r c ≤ 20) giving the number of rows and columns of the crossword. This will be followed *r* rows each containing c characters. Each character will be either an uppercase letter or a '@' representing a black square. The file will be terminated by a pair of zeroes (0 0).

For each crossword, the output will be a form showing the clue number and length. Follow the format shown in the example. Leave one blank line between successive crosswords.

Sample Input

```
5 13
FIRST@SECOND@
@@O@I@E@@@@@@
@@V@MOTOR@@@@
@@E@E@@U@@@@@
@@R@STUTTER@@
0 0
```

Sample Output

```
Across
1.  (5)
4.  (6)
5.  (5)
7.  (7)
Down
2.  (5)
3.  (5)
4.  (3)
6.  (3)
```

Problem I Molecular Synthesis 30 Points

Organic molecules can be amazingly complex and need a great variety of shapes and conventions to represent them, particularly if we wish to depict details of their 3-dimensional structures. However, if we restrict ourselves to reasonably simple compounds, i.e. those with only single bonds between atoms, then we can represent them on a simple rectangular grid with bonds aligned horizontally or vertically. In such a molecule, carbon is bonded to four adjacent atoms, nitrogen to 3, oxygen to 2 and hydrogen to 1. Unfortunately not all such grids represent valid molecules. Your task is to write a program that will determine whether a given grid represents a valid molecule.

Input will consist of a series of possible molecules portrayed as grids. The first line of the input for each molecule will consist of a pair of integers (*r* and *c*, $1 \le r\ c \le 5$) representing the number of rows and columns in the rectangle to follow. The next *r* lines will contain *c* characters each, where the characters are chosen from the set {'`.`' (empty), '`H`' (hydrogen), '`O`' (oxygen), '`N`' (nitrogen), '`C`' (carbon)}. The file will be terminated by a line containing two zeroes (0 0). Note that 'molecules' classified as valid may not be physically realisable, and that there may in fact be more than one molecule present.

For each potential molecule in the input, output one of the following lines:

```
Molecule <num> is valid.
Molecule <num> is invalid.
```

where <num> is a running number starting at 1.

Sample Input

```
3 4
HOH.
NCOH
OO..
3 4
HOH.
NCOH
OONH
4 10
OOOOOOOOOO
OOOOOOOOOO
OOOONOOOOO
OOOOOOOOOO
2 3
HOH
HOH
0 0
```

Sample OutPut

```
Molecule 1 is valid.
Molecule 2 is invalid.
Molecule 3 is invalid.
Molecule 4 is valid.
```

Problem K Molecular Synthesis 100 Points

The specifications of the 100 point version are the same as that of the 30 point version, except that the limits on r and c are $1 \le r\ c \le 20$. Any program submitted for the 100 point version will also be tested against the test data for the 30 point version, and may earn credit for the latter even if it fails on the 100 point test data.

A maximum of 100 points is available for the two versions of the problem.

Problem L Fragment reassembly 100 Points

The Government, fearing that their secret plans to turn the University into a theme park might be discovered, have put the file through an electronic shredder, which has chopped the text up into overlapping pieces. Your mission, should you choose to accept it, is to write a program which can read a list of text fragments and use the overlaps to reassemble them so that we can reveal the Governments plans.

Input will consist of a series of problems. Each problem will consist of 1..20 lines of text terminated by a line containing a single #. Each line will contain between 1 and 72 characters. A word is deemed to be a sequence of 1 or more printable characters, and words will be separated by exactly one space. The sequence of problems will be terminated by a line containing a single #. The # character will not occur anywhere else in the file other than the specified places.

For each problem the output will consist of any valid arrangement of the fragments that includes every fragment and allows for all indicated overlaps, i.e. no segments are repeated. Solutions are not necessarily unique and no minimality criterion should be applied. We will accept any text that satisfies the above criteria.

For each problem output the recreated text as a sequence of one or more lines, where each line contains no more than 72 characters. Do not break a line other than in the space between words, in which case do not output the space. After each problem's answer there should be a line consisting of a single '#'.
<<Why? Why not just a blank line?>>

Sample Input

```
they chose Avant
from the regular text.
For headings, they
stands out nicely from
sans-serif font that stands
Avant Garde, a sans-serif
from the regular text.
#
a b r a
c a d a b r a
#
a b r a
c a d a b r a
r a c
#
#
```

Sample output

```
For headings, they chose Avant Garde, a sans-serif font that stands out
nicely from the regular text.
#
c a d a b r a
#
c a d a b r a c (Alternatively a b r a c a d a b r a)
```

(This from Richard: How do people feel about the non-uniqueness of answers? It doesn't bother me, because the check is fairly simple, and I am happy to supply a checking program.)

Problem M Data Mining 100 Points

Whenever you make a purchase at the New World Order supermarket, a copy of your docket is sent to the Master Control Program. They want to know all about shoppers' buying habits.

What the Master Control Program gets is a sequence of itemsets. An itemset is a set of product codes. For example, it might get 128 92 47 638, which might mean Wine, Cheese, Biscuits, and Pâté. What the Master Control Program does with this is to look for subsets of items that are commonly bought together. For example, Cheese and Biscuits might often be bought together. What the New World Order supermarket does with this information is a matter for conjecture and dread.

Write a program that will determine, for a given threshold *n* and a set of itemsets such as the one above, what subsets of the items occur in *n* or more of the itemsets.

Input will consist of the integer *n* ($1 \leq n \leq 1000$) on a line by itself. This will be followed by a sequence of itemsets, terminated by a single 0 on a line by itself. Each itemset is a list of 1 to 20 positive integers. The elements of a dataset will be distinct, but not necessarily ordered.
<<Do we want multiple input sets, terminated by n = 0? >>
Output will consist of the number of common subsets that were found, followed by descriptions of these subsets in lexicographic order. The first line of the output will be either be the line: "`1 common subset found.`" if only one subset was found, or the line "`k common subsets found.`" if k (k > 1) were found. The description of a subset consists of its elements in increasing order, with no leading zeros and separated by single spaces followed by a space, an opening parenthesis ('('), the number of times that itemset occurred and a closing parenthesis (')'). Follow the examples shown.

Sample Input

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

Sample Output

```
5 common patterns found.
1 4 (2)
1 5 (2)
3 4 (3)
3 9 (3)
4 9 (2)
```

Method

The most efficient method I know works bottom-up, starting with singletons. You get level k+1 from level k by looking for pairs A...XY and A...XZ and combining them to A....XYZ if that combination is common enough.

Problem N Protein Similarity 100 Points

Proteins are very big molecules, made of smaller molecules called amino acids. The sequence of amino acids is called the primary structure of a protein; there are also secondary and tertiary structures describing how it folds up into a 'molecular machine'.

There are 20 different amino acids used by most organisms. Biochemists abbreviate their names to single letters: A=Alanine, C=Cysteine, D=Aspartic acid, E=Glutamic acid, F=Phenylalanine, G=Glycine, H=Histidine, I=Isoleucine, K=Lysine, L=Leucine, M=Methionine, N=Asparagine, P=Proline, Q=Glutamine, R=Arginine, S=Serine, T=Threonine, V=Valine, W=Tryptophan, and Y=Tryosine. They also use three more letters when writing down what they know about a protein: X means no idea which, B means either
Asparagine or Aspartic acid, and Z means either Glutamine or Glutamic acid.

Sometimes we have two proteins, and want to know how similar they are. A mutation might add an amino acid, delete one, or change one into another. The similarity matrix tells us about the cost of changing amino acid into another (+ve means low cost, -ve means high cost). In addition we need a table that tells us how costly it is to add or delete a particular amino acid. Your task is to write a program which reads pairs of proteins and reports their similarity scores.

The input consists of a similarity matrix, an indel cost table, and a series of protein pairs terminated by a line containing a single #.

Since similarity matrices are symmetric, there is no point entering all the numbers. A similarity matrix will be presented as 23 lines. The first line will have one number, sim[1,1]. The second line will have two numbers, sim[2,1]=sim[1,2], and sim[2,2]. The third line will have three numbers, sim[3,1]=sim[1,3], sim[3,2]=sim[2,3], and sim[3,3]. And so it goes, up to the 23rd line, which will have 23 numbers. All of these numbers will be integers between -999 and 999 inclusive. There will be spaces between them, and may be spaces before and/or after them. The entries are presented in the order A B C D E F G H I K L M N P Q R S T V W X Y Z.

The indel cost table will be presented as one line with 23 numbers. All these numbers will be non-negative integers, which is the opposite of the convention used in the similarity matrix. The same order is used here as for the similarity matrix.

A protein will be represented by 0 or more lines each containing exactly 60 amino acid letters, followed by one line containing 0..59 amino acid letters. Each protein will contain at least one amino acid. For this problem, no protein will have more than 400 amino acids.

For each protein pair in the input, there is to be an output line of the form "Similarity for pair *n* is *score*.", where n is a counter starting at 1 and *score* is the highest score obtainable for that pair of proteins.

Sample input

```
6
-5 6
-8 -14 10
-4 6 -16 8
-3 0 -16 2 8
-9 -12 -15 -17 -16 9
-3 -4 -11 -4 -5 -10 7
-8 -2 -8 -5 -6 -7 -10 9
-6 -7 -7 -9 -6 -3 -13 -11 9
-8 -3 -16 -6 -5 -16 -8 -8 -7 7
-7 -10 -17 -15 -10 -4 -12 -7 -2 -9 7
-6 -12 -16 -13 -8 -5 -10 -13 -2 -3 0 11
-5 6 -13 1 -3 -10 -4 -1 -6 -2 -8 -11 8
-2 -8 -9 -9 -7 -11 -7 -5 -10 -8 -8 -9 -7 8
-5 -4 -16 -4 0 -15 -8 0 -9 -4 -6 -5 -5 -4 9
-8 -9 -9 -12 -11 -10 -11 -3 -6 -1 -10 -5 -7 -5 -2 9
-1 -2 -4 -5 -5 -7 -3 -7 -8 -5 -9 -6 -1 -3 -6 -4 7
-1 -4 -9 -6 -7 -10 -7 -8 -3 -4 -8 -5 -3 -5 -7 -8 0 7
-3 -9 -7 -9 -8 -9 -7 -7 1 -10 -3  -2  -9  -7  -8  -9  -8  -4  7
-16 -11 -18 -17 -19  -6 -17  -8 -16 -14  -7 -15  -9 -16 -15  -3  -6 -15 -18 13
-4  -6 -11  -7  -6  -9  -6  -6  -6  -6  -7  -6  -4  -6  -6  -7  -4  -5  -6 -13  -6
-9 -7 -5 -13  -9  1 -16  -4  -7 -10  -8 -13  -5 -16 -14 -11  -8  -7  -8  -6  -9 10
-4  -1 -16 0 6 -16 -6 -2  -7  -5  -8  -6  -4  -5  7  -5  -6  -7  -8 -17  -6 -11  6
13 10 20 14 13 23 13 18 16 16 12 30 7 17 21 19 18 9 29 2 23 13 8
ILHXAPDIRBXDDNPPGIGTGBKWYRWALNVGZHPQEXHDHIQTPNLRHYIYAMGAPEXB
FGXMSGWRLZTIFIGRSPVIVWEYKMBYXTKRT
MFXNAWAVFHZYVM
GHGEABFKSFMBKMFFYIEEQBVALEYXLXRYLKXRWTQRPGYRBIQDIQAWBTF
HEQFFFYDGNHWGRWXQBXFQWWITCWNWHRHXZCZBQAINGNPVLALGGEAWPCSC
SVWFFHINGKXBGKNHVLCGYZHXFITLXKPZIKEEHLBRBNHGMBZSVDSSWQYMNAPB
PKXHHWSCHKZQWLVBDNYXIKWGCPTDVTK
KANSFRTTCIYTGZKAPATDIFPEVTLGWLWKHYFIKMXN
#
```

Sample output

```
Similarity for pair 1 is -11398.
Similarity for pair 2 is 5262.
Similarity for pair 3 is -367.
```

<<This is what I was given. I think that the lines for the proteins have been broken in funny places, but I don't have time to fix it now.>>