Problem A: Code Breaking

Periodic permutation is a simple encryption technique which involves choosing a period, k, and a permutation of the first k numbers. To encrypt a message, split the message into groups of k characters (padding if necessary) and apply the given permutation. Decryption involves taking groups of k characters and performing the inverse permutation. Thus for $k = 4$, a permutation could be 2431. This would encrypt 'Mary' to 'yMra' and 'Maryan' to 'yMra?a?n'. Once one knows the permutation, one can apply its inverse to other encrypted messages (cyphertext) to recover the original text (plaintext).

Write a program that will read (plaintext, cyphertext1, cyphertext2) triples, and for each (plaintext, cyphertext1) pair determine whether or not a periodic permutation encryption method has been used. If it has, determine the value of k and the permutation function and apply the reverse permutation to cyphertext2 to recover the corresponding plaintext.

Input will consist of a series of (plaintext, cyphertext1, cyphertext2) triples. Lines will be no more than 80 characters long. The first two strings (of length n) represent the first n characters of the plaintext and cyphertext. There is no implication that n is a multiple of k. The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each triple in the input. If a permutation cycle has been found, apply the inverse permutation to cyphertext2, padding it if necessary with '?'. If no periodic permutation can be found (with period less than or equal to the length of the plain and cyphertext1 strings) that transforms the plaintext into the cyphertext, then print cyphertext2 unchanged. If more than one periodic permutation could have mapped the plain text to the cyphertext1, then apply the periodic permutation that has the smallest value for k. There will never be more than one shortest permutation function that matches the data.

Sample input

```
Mary had a little lamb!!
aMyrh daa l tilt ealbm!!
hTsii  s aetts
Foobar
blargg
No cycle
abc
bca
abcd
#
```

Sample output

```
This is a test
No cycle
```

Problem B: Eeny Meeny

In darkest <name of continent/island deleted to prevent offence> lived a tribe called the "Eeny Meenys". They got this name from their way of choosing a chief for a year. It appears that a newspaper reporter visited the tribe and managed to get across a few ideas of civilisation, but apparently came to an unfortunate end before finishing the job. Thus the tribe no longer had a permanent chief; the chief's term was exactly one year. At the end of that time, they ate the current chief, and chose another chief. Their method of choosing a chief was the "Eeny meeny miny mo" method. All eligible tribal members (women were also eligible—one of the blessings of civilisation the tribe had adopted) stood in a circle, a starting place was chosen, and the chief medicine man (who was ineligible for chieftainship) went around counting out 'E', 'e', 'n', 'y', 'M', 'e', 'e', 'n', 'y', 'M','i', 'n', 'y', 'M', 'o!', 'E', 'e', 'n', 'y', 'M', 'e', 'e', 'n', 'y', 'M', 'i', 'n', 'y', 'M', 'o!', ... At every 'o!', the person indicated was pushed out of the circle which then closed up and the count restarted with his neighbour (the one who would have been 'E' anyway). This process continued until only one was left—the new chief.

While the chance of glory for a year makes the job of chief highly attractive to tribal members, you (possessing a computer decades before they were invented) find the brevity of the glory unappealing. You have managed to find out that the count this year will start with Mxgobgwq (a very large person), so you would like to know where not to stand. You don't know the direction, nor how many eligible people there are, but you can estimate the number (it is certainly less than 500).

Write a program that will determine the 'first' (i.e. closest to Mxgobgwq) safe position to stand, regardless of the actual number of people and the direction of count (clockwise or anti-clockwise).

Input will consist of a series of lines, each line containing the upper and lower estimates of the number of eligible people (both numbers inclusive). The file will be terminated by a line containing two zeroes (0 0).

Output will consist of a series of lines, one for each line of the input. Each line will consist of a single number giving the number of the position closest to Mxgobgwq that will not be chosen as chief for any number in the given range and for either direction of elimination. If no position is safe then print `"Better estimate needed"`.

Sample input

```
80 150
40 150
0 0
```

Sample output

```
1
Better estimate needed
```

Problem C: Hearts

There are 52 playing cards in a pack, divided into suits, and, within suits, into denominations. The suits are (in order, lowest to highest) Clubs, Diamonds, Hearts and Spades, abbreviated C, D, H and S. The 13 denominations (or face values) are (from lowest to highest): 2, 3, 4, 5, 6, 7, 8, 9, 10 (T), Jack (J), Queen (Q), King (K) and Ace(A). A higher card will beat a lower card in the same suit, but will not usually beat any other card in a different suit. An exception to this is the 'trump' suit—if a suit is designated to be a trump suit (by whatever means the rules of the game allow), then **any** card of that suit will beat **any** card of any other suit.

A simplified version of an old card game called Hearts is played as follows. The dealer deals cards clockwise, one by one, face downward, to four other players and himself, starting with the player on his left, who thus gets the first card, followed by the sixth, and so on, while the dealer gets the fifth card, followed by the tenth, and so on. When each player has 10 cards there will be two left—these are exposed and the suit of the one of higher denomination determines the trump suit. If there is a tie, then the highest ranking suit becomes the trump suit.

A 'game' consists of 10 'tricks', each containing 5 cards, one from each player. For each trick, one player 'leads', i.e. plays a card face up on the table, the rest of the players then 'follow', in clockwise order. The player to the dealer's left leads to the first trick, thereafter the winner of each trick leads to the next trick. A player must follow suit if possible, i.e. play a card of the same suit as the one lead. If he cannot, then he must trump it (play a card of the designated trump suit). If he cannot trump it (because he has no cards in the trump suit), he discards a card. If a trick is trumped, then the person playing the highest trump wins the trick, otherwise the person playing the highest card of the correct suit wins it.

Strategies are as follows:

1. Leader: The leader always plays the highest card in his hand. If there is a tie and one of the cards is a trump card, then he leads the trump, otherwise he plays the highest ranking suit.

2. Follower: If possible he must play the highest card in his hand of the correct suit. If he has no cards in that suit then he plays the highest trump he has. If he cannot trump it he plays the highest card in his hand, breaking ties as previously specified.

When all the tricks have been played, each player examines the tricks he has taken and scores the face value of any Heart he has (Jack counts 11, Queen counts 12, King counts 13 and Ace counts 14). This score is recorded.

Write a program to simulate the playing of this game.

Input will consist of a series of decks of cards, each deck spread over four lines as shown below. The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each deck in the input. Each line will consist of 5 numbers reflecting the scores of the individual players, starting with the dealer and proceeding clockwise through the rest of the players. Each score will consist of a number right justified in a field of width 3.

Sample input

```
TS QC 8S 8D QH 2D 3H KH 9H 2H TH KS KC
9D JH 7H JD 2S QS TD 2C 4H 5H AD 4D 5D
6D 4S 9S 5S 7S JS 8H 3D 8C 3S 4C 6S 9C
AS 7C AH 6H KD JC 7D AC 5C TC QD 6C 3C
#
```

Sample output

```
 22  0 68  0 14
```

Problem D: Bonus Bonds

The government of Impecunia does not levy any taxes, instead it raises money by the (sometimes forced) sale of Bonus Bonds. Originally the Bonds were numbered using a 7 digit number prefixed by a one digit code in the range 1 to 9 representing the region of Impecunia in which the bond was sold. However the scheme has proved so popular that the numbering scheme has been extended by a further two digits. To retain compatibility with the previous scheme, the 8th digit from the right (the third from the left) still designates the region of sale. At the same time, a 'central' region was created and has been given the designation 0. For security reasons no bond may be numbered with a number consisting entirely of zeroes, thus, although the original bonds all started from zero (since the region code was non-zero), the bonds from the central region start from 0000000001.

Each month, the winning numbers are drawn for each region independently. The equipment generates a stream of single digits and it would appear to be simple enough to collect these together in groups of ten and compare the results with the list of Bond-holders. However, the equipment is a little antiquated and is liable to various breakdowns, thus it is desirable to only generate numbers that lie within the allocated range and with the same distributions of digits at each position as would be found by examining all the bonds sold for that region. Thus if we wish to draw N numbers for a given region, the equipment is set to generate 10 streams of N digits, one for each position. The first winning number is then made up by taking the first digit from each stream, the second winning number is composed of the second digit in each stream, and so on. For each stream, the equipment is adjusted so that the distribution of digits it generates closely matches the actual distributions of digits in that position on the allocated Bonds. The state auditors generate a table of these distributions so that the two may be compared.

Write a program that will generate the table for the state auditors for any given draw. For each region, the program will read the serial number of the **next bond to be sold in that region** so that it can calculate the distributions. Since the output is voluminous, your program will only need to print the digit distribution for a particular digit position.

Input will consist of a series of lines, each line consisting of a ten digit number representing the **next** bond number to be sold in a particular region and an integer in the range 1 to 10 representing the desired character position. It is possible that some regions will appear more than once in the input stream, and that others will not appear at all. The file will be terminated by a line consisting of 0000000000 0.

Output will consist of a series of tables, one for each line of the input. Each table will consist of ten rows, one for each digit in the range 0 to 9. Each row will consist of a single number giving the numbers of times that digit appears in the sequence numbers at the desired position. Each number will be right justified in a field of width 11. Separate tables by one blank line.

Sample input

```
4810000000 1
0000000000 0
```

Sample output

```
 100000000
 100000000
 100000000
 100000000
  80000000
         0
         0
         0
         0
         0
```

Problem E: Bit Maps

The bitmap is a data structure that arises in many areas of computing. In the area of graphics, for example, a bitmap can represent an image by having a 1 represent a black pixel and a 0 represent a white pixel.

Consider the following two ways of representing a rectangular bit map. In the first, it is simply represented as a two dimensional array of 1s and 0s. The second is based on a decomposition technique. First, the entire bit map is considered. If all bits within it are 1, a 1 is output. If all bits within it are 0, a 0 is output. Otherwise, a D is output, the bit map is divided into quarters (as described below), and each of those is processed in the same way as the original bit map. The quarters are processed in top left, top right, bottom left, bottom right order. Where a bit map being divided has an even number of rows and an even number of columns, all quarters have the same dimensions. Where the number of columns is odd, the left quarters have one more column than the right. Where the number of rows is odd the top quarters have one more row than the bottom. Note that if a region having only one row or one column is divided then two halves result, with the top half processed before the bottom where a single column is divided, and the left half before the right if a single row is divided.

Write a program that will read in bitmaps of either form and transform them to the other form.

Input will consist of a series of bit maps. Each bit map begins with a line giving its format ("B" or "D") and its dimensions (rows and columns). Neither dimension will be greater than 200. There will be at least one space between each of the items of information. Following this line will be one or more lines containing the sequence of "1", "0" and "D" characters that represent the bit map, with no intervening spaces. Each line (except the last, which may be shorter) will contain 50 characters. A "B" type bitmap will be written left to right, top to bottom. The file will be terminated by a line consisting of a single #.

Output will consist of a series of bitmaps, one for each bit map of the input. Output of each bit map begins on a new line and will be in the same format as the input. The width and height are to be output right justified in fields of width four.

Sample input

```
B 3  4
001000011011
D  2   3
DD10111
#
```

Sample output

```
D   3   4
D0D1001D101
B   2   3
101111
```

Problem F: Laser Lines

A computer chip manufacturer has discovered a new way to combine opto-electronics and ordinary electronics by forming light-emitting and receiving nodes on the surface of the chip. These can be used to send messages to each other in a direct line-of-sight manner, thereby speeding up operation considerably by allowing a much greater density of information transfer. One difficulty is that the nodes must all be able to send messages to each other; no node should block the line-of-sight between two other nodes. The manufacturing method ensures that the nodes will be positioned exactly on the points of a lattice covering the chip, so their coordinates are given by integers between 0 and 9999 (inclusive) except that for technical reasons no node can appear at point (0, 0).

Write a program that will read in sets of coordinates of these nodes and determine whether any of them lie on lines containing three or more nodes. Because of the layout method used, it is envisaged that there may well be several lines containing three nodes, but that 'longer' lines will be increasingly rare. However, no line will contain more than 10 points.

Input will consist of a series of data sets, each set containing the coordinates of between 3 and 300 points (both inclusive). Each set will start on a new line. The coordinates will be pairs of integers in the range 0 to 9999 and each set will be terminated by a pair of zeroes (0 0). Successive numbers will be separated by one or more spaces; in addition a data set may be split into several lines, such splits will only occur between coordinate pairs and never between the elements of a coordinate pair. The entire file will also be terminated by a pair of zeroes (0 0). Note that there will be several test cases, but only one will contain more than 100 points.

Output, for each set, is either the message `"No lines were found"`, or the message `"The following lines were found:"`, followed by the sets of points lying on straight lines, each set ordered first by x, and if the x's are equal, then by y. All coordinates are in a field of width 4, and are separated by a comma; the points are delimited by brackets, with no spaces between successive points. The lines themselves are ordered in a similar manner to the points on each line; i.e. by considering the first point on each line, and if more than one line starts at that point, by considering the second point on the line.

Sample input

```
  5 5 8 7 14 11 4 8   20 15
12 6  18 21 0  0
5 5 8 8 14 13 0 0
5 5 25 17 20 23 10 11 20 14 15 11 0 0
0 0
```

Sample output

```
The following lines were found:
(   4,   8)(   8,   7)(  12,   6)
(   5,   5)(   8,   7)(  14,  11)(  20,  15)
```

```
(  12,   6)(  14,  11)(  18,  21)
No lines were found
The following lines were found:
(   5,   5)(  10,  11)(  20,  23)
(   5,   5)(  15,  11)(  20,  14)(  25,  17)
```

Problem G: Roman Numerals

The original system of writing numbers used by the early Romans was simple but cumbersome. Various letters were used to represent important numbers, and these were then strung together to represent other numbers with the values decreasing monotonically from left to right. The letters they used and the numbers that were represented are given in the following table.

I	1	V	5
X	10	L	50
C	100	D	500
M	1000		

Thus 1993 was written as MDCCCCLXXXXIII. This system was then superseded by a partially place-oriented system, whereby if the above rule of decreasing values was broken, it meant that the immediately preceding (lower) value was deemed to be negative and was subtracted from the higher (out of place) value. In this system 1993 was usually written as MCMXCIII. There is still some controversy as to which letters could precede which other letters, but for the purposes of this problem we will assume the following restrictions:

1. A letter from the left column can never appear more than three times in a row, and there can never be more than one other occurrence of that letter.

2. A letter from the right column can never appear more than once.

3. Once a letter has been used in a 'negative' position, all subsequent characters (apart from the one immediately following) may not be greater than that character.

Thus we could write MXMIII for 1993 or CCXCIV for 294, however we could not write ILV for 54, nor could we write LIL for 99. Note that 299 could be written as CCXCIX or CCIC

Given a Roman sum, we can either interpret it as such or as an encoding of an Arabic sum. Thus V+V=X could be interpreted as an ambiguous encoding of an Arabic sum with V $\in$ {1, 2, 3, 4} and X = 2 * V. Similarly, X+X=XX could be interpreted as a correct Roman sum but an impossible Arabic encoding (apart from the trivial encoding X = 0) and XX+XX=MXC as an incorrect Roman sum, but a valid encoding with M = 1, X = 9, and C = 8.

Write a program that will read in sums in Roman numerals and determine whether or not they are correct as Roman sums and also whether they are impossible, ambiguous or valid as Arabic encodings. Assume that zero will never appear on its own or as a leading digit, and that no two Roman numerals map onto the same Arabic digit.

Input will consist of a series of lines, each line consisting of an apparent Roman sum, i.e. a valid Roman number, a plus sign (+), another valid Roman number, an equal sign (=) and another valid Roman number. No Roman

number will contain more than 9 letters. The file will be terminated by a line consisting of a single #.

Output will consist of a series of lines, one for each line of the input, and each containing two words. The first word will be one of (Correct, Incorrect) depending on whether the Roman sum is or is not correct. The second word will be separated from the first by exactly one space and will be one of the set (impossible, ambiguous, valid) depending on the Arabic sum.

Sample input

```
V+V=X
X+X=XX
XX+XX=MXC
#
```

Sample output

```
Correct ambiguous
Correct impossible
Incorrect valid
```

Problem H Arithmoglyphics

The firm for which you work prides itself on producing calculators to satisfy every conceivable desire. Their greatest triumph to date has been an electronic abacus, which shows beads moving at lightning speed on the screen. Your boss has just heard about ancient Egyptian mathematics and has told you to write a program for a feasibility study . At first you think this will be an easy modification of the popular Roman Numerals calculator, but then you discover how the Egyptians handled fractions. They had not invented zero, or negative or irrational numbers and were ambivalent about division. This led them to reject fractions such as 3/7, but since they couldn't deny that things can be divided into 7 equal parts, they were happy enough about 1/7. They also accepted 2/3, which they thought of as 1–1/3. To avoid getting ridiculous numbers such as 1/7+1/7+1/7 when three 'seventh-parts' were added together (this is just a disguised form of the rejected 3/7), whenever fractional numbers were combined the result was written as a sum of fractions of the form 1/x (possibly including 2/3), with all values of x **different** . Thus 1/7 + 1/7 + 1/7 was written as 1/4 + 1/6 + 1/84. It is easy to see that every fraction can always be written in this way, and in general there are many possibilities; for example, we have the equalities: 1/12 + 1/15 = 1/10 + 1/20 = 1/8 + 1/60 + 1/120 = 1/7 + 1/140.

From now on we will use the notation $(a_1,a_2,a_3,\ldots)$ to mean $1/a_1 + 1/a_2 + 1/a_3 + \ldots = \Sigma\ 1/a_i$. The ai will all be different and in increasing order. We will use the number 1 to represent 2/3 (since 1/1 is not a fraction) — thus (1,6,13) means 2/3 + 1/6 + 1/13.

Three rules were observed when the result of a computation (sum, difference or product) on fractions was rewritten as a single fraction:

(a) The fewest possible numbers were used in the sum (thus (8) + (60) + (120) would be written as (12,15) or (10,20) rather than (8,60,120))
(b) Among the possible shortest representations the one coming first in dictionary order would be used. Thus the correct form of (8) + (60) + (120) is (7, 140)
(c) Since the Egyptian numeral system did not lend itself to large numbers, no number greater than or equal to 1,000,000 was used in a fraction.

A list of integers of the form (a,b,c,..) such that 1/a + 1/b + 1/c +... < 1 and satisfying the rules above, will be called an Egyptian fraction. Write a program which will perform calculations on Egyptian fractions.

Input will be from a file called PROBLEMH.DAT and will consist of a series of lines, each line consisting of two Egyptian fractions, separated by a +, – or * (for sum, difference and product), possibly with one or more blanks. Each Egyptian fraction will be written with an open bracket, a list of whole numbers in increasing order separated from each other by commas and possibly blanks, and terminated by a close bracket. The file will be terminated by a line consisting of a single empty Egyptian fraction ().

Output will consist of a series of lines, one for each line of the input. Each line will consist of the result of the input expression written as an Egyptian fraction if the answer is less than 1, as a single 1 if the answer is exactly 1, or as '1+' followed by an Egyptian fraction if the answer is more than 1. If the answer less than or equal to zero the output line must contain the word 'Invalid'. There must be no blanks in the output line.

Example

INPUT	OUTPUT
(1) - (3)	(3)
(2) + (2)	1
(1, 6) + (1, 6)	1+(1)
(13,999983) - (13,999983)	Invalid
(1) * (3,8,120)	(5,9)
()	