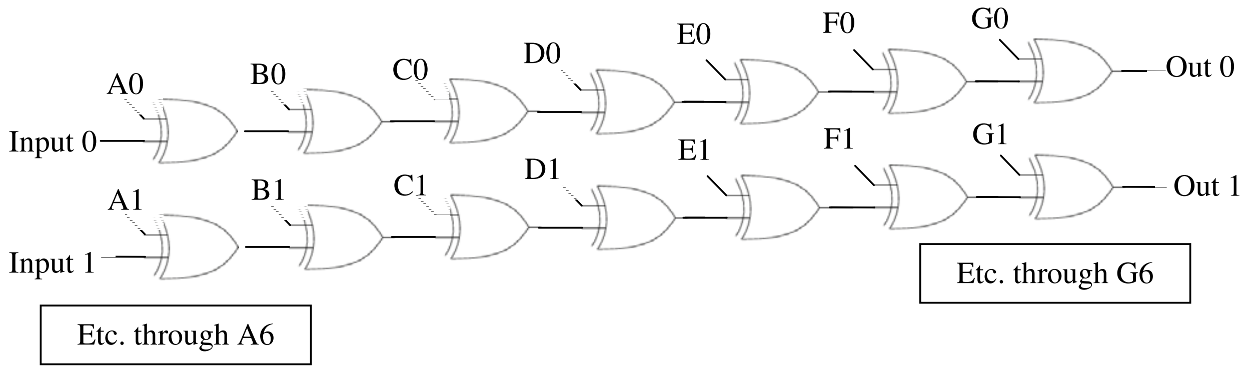
Problem 1: Secret Decoder Ring

Consider the following circuit diagram made up of exclusive-or gates:



Obviously output zero is determined by not only input zero, but also the setting of A0, B0, C0, … Let's call each association between input N and output N a "row". Suppose that there are seven such rows, so that "A0" is the upper left external control, "G0" is the upper right, "A6" is the lower left, "G6" is the lower right. One can then specify, in a 7×7 matrix, the setting of all A0..G6, as in:

1	0	0	1	0	0	1
1	1	1	1	0	0	0
1	1	0	0	0	0	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
1	1	1	0	0	0	0
0	0	0	0	0	0	0

Note that in the case of row six, which is all zeros, this yields an identity function where output six will always be equal to input six.

This matrix and logic can be used as a primitive encoding/decoding device. Take an input character and pass it to the logic diagram with character bit zero going to input zero, bit one going to input one, and so on. Place output zero in bit zero of a new character; place output one in bit one, and so forth. The output will be another character, likely different from the input character.

Example

An easy example can be constructed using the following table:

0	0	0	0	0	0	1
0	0	0	0	0	0	0
0	0	0	0	0	0	1
0	0	0	0	0	0	0
0	0	0	0	0	0	1
0	0	0	0	0	0	0
0	0	0	0	0	0	0

In this case bits zero, two, and four will be inverted. Inserting a letter 'A', which is ASCII 0x41 (hexadecimal) or binary 01000001 will generate 01010100 or 0x54, which is 'T'. This completes the processing of one character by the Secret Decoder Ring.

After each character is completed, consider each row in the table as an integer. Row zero in the above example has a value of one. Add one to each row after processing a character. Subsequent decoder tables would thus contain:

0	0	0	0	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	0	0	1
0	0	0	0	0	0	1

After 1st character

0	0	0	0	0	1	1
0	0	0	0	0	1	0
0	0	0	0	0	1	1
0	0	0	0	0	1	0
0	0	0	0	0	1	1
0	0	0	0	0	1	0
0	0	0	0	0	1	0

After 2nd character

0	0	0	0	1	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	0
0	0	0	0	0	1	1
0	0	0	0	0	1	1

After 3rd character

Continue this process until the end of the data is encountered (see below) for each table. A row in a table will never overflow a seven bit value, so there is no need to test for this situation.

Input

There will be multiple tables and multiple input characters to be decoded using each table. A table will appear first, specified using 7 input lines each containing 7 integers, each of which is either 0 or 1. There is a blank between each pair of bits. Each row is arranged as described above – bit six, then bit five, and so forth down to bit zero. Following the table are additional lines of data, each line representing one character to be decoded. These lines have the same format as the table lines. The input line for the last character to be decoded by a table is followed by a line containing 7 zeroes. This input sequence may then repeat with additional tables and sets of characters to be decoded. The last repetition (a table and characters to be decoded) is followed by a line containing 7 zeroes. (This means no table will ever have 7 zeroes in its first row, and no character to be decoded will consist of 7 zeroes.)

Output

For each case, display the table number (1, 2, …) and the decoded characters. Display a blank line after the output for each table. Use the format shown in the sample below. There will not be any "non-printable characters" in the output.

Sample Input

```
0 0 0 0 0 0 1
0 0 0 0 0 0 0
0 0 0 0 0 0 1
0 0 0 0 0 0 0
0 0 0 0 0 0 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
1 0 1 0 1 0 0
0 1 1 1 1 0 1
0 1 0 1 0 0 1
1 0 1 0 0 0 1
0 1 0 1 1 1 1
1 0 0 0 1 1 0
1 0 1 0 0 1 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

Output for the Sample Input

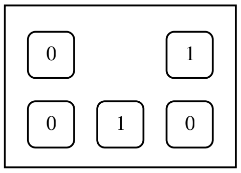
```
Table 1: ABCDEFG
```

Problem 2: Do the Binary Shuffle

Almost everyone is familiar with the "seven puzzle." The puzzle board has M rows and N columns of squares, each square (except one) containing a tile marked with a letter or digit or symbol, and one empty square. In each move, any one of the tiles neighboring the empty square can be moved (by sliding it) from its square into the empty square giving a different puzzle configuration.

In this problem each of the tiles in such a puzzle is marked with a binary digit (0 or 1). After you make exactly K moves of the tiles, treat each row as a separate N-digit binary number (treating the empty location as if it contained zero). What we want to know is the maximum number of unique binary integers that can possibly be created in this manner, considering all possible sequences of K moves.

Let's consider a simple example. Suppose the initial puzzle configuration looked like that shown below, and that you make exactly one move. There are three possible choices for that move. The first choice is to move the 0 on the first row right; this yields the binary numbers 001 and 010 (treating the empty square as if it contained zero). The second choice is to move the 1 on the first row left. This yields the binary number 010 on each row. The third choice is to move the 1 on the second row up, yielding 011 and 000. So making one move could yield at most four binary numbers: 001, 010, 011 and 000.



Input

There will be multiple cases numbered 1, 2, …. The input for each case begins with a line containing three integers M, N, and K. M specifies the number of rows in the puzzle (between 1 and 5), and N specifies the number of columns (between 1 and 6). K specifies the maximum number of moves you may make; it will be no larger than 15.

Following the first line of input there will be M lines, each containing N characters chosen from '0', '1', and 'X'. Each character may be preceded and followed by whitespace (blanks or tabs). These lines specify the binary values on each tile and the position of the empty square (by the 'X' character). 'X' will appear exactly once.

The last case will be followed by a line containing a single 0.

Output

For each case, display the case number (1, 2, …) and the maximum number of different binary values that could be produced. Use the format shown in the samples below. Display a blank line after the output for each case.

Sample Input

```
2 3 1
0 X 1
0 1 0
2 2 2
0 X
1 0
0
```

Output for the Sample Input

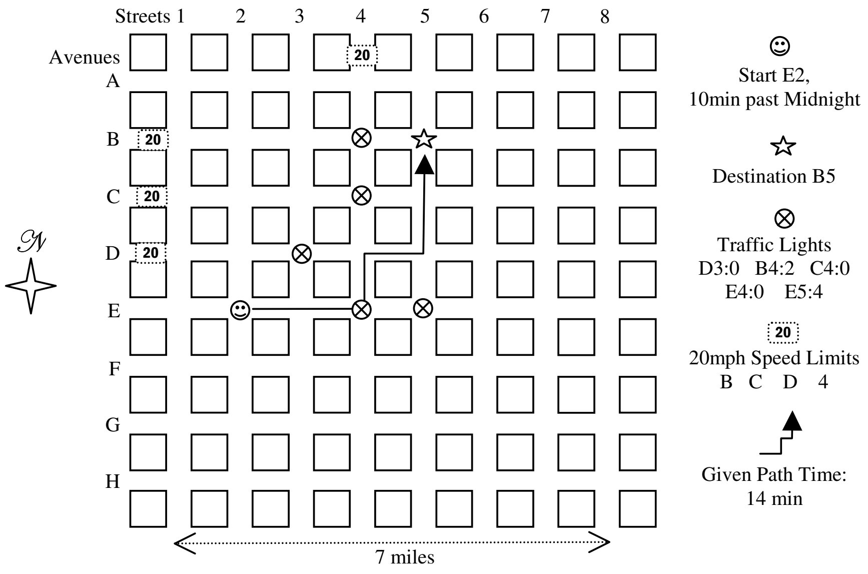
```
Case 1: 4 binary numbers

Case 2: 3 binary numbers
```

Problem 3: The Octavia Traffic Lights

The City of Octavia is one of the most orderly and well-organized cities in the world. The traffic in Octavia flows on 8 major avenues, named A through H and 8 major streets, numbered 1 through 8, all of which are two-way. The avenues stretch in the East/West direction and are perpendicular to the streets, which go North/South. Each street and each avenue has a speed limit – either 20 or 30 miles per hour (Octavia is not known to be fast-paced). The city infrastructure is so well designed that the distance between any two consecutive streets or avenues is exactly 1 mile. Thus, Octavians live in large city blocks that are each one square mile each (as illustrated in the diagram below), but drive only on the 8 streets and 8 avenues. Each intersection in Octavia is labeled with the name of the avenue and street that meet at that point; for example H8 is in the southeast corner of the city, while D5 is near the center.

The Octavia City Council is conducting a traffic study and wants to install traffic lights on some (or all) intersections in Octavia. The city has already contracted Imperial Light Machines (ILM) to deliver the traffic lights. The ILM traffic lights are very simple: they only have red and green signals and are designed to cycle from green to red and back to green every 4 minutes, showing green for 2 minutes, then showing red for another 2 minutes. While this is adjustable, the Octavian Traffic Committee (OTC) has determined that 4 minutes is precisely the cycle time that they would need to control all the traffic in the city. Each traffic light is reset at a particular time every 24 hours and starts alternating from that point on, giving a green light to the North/South direction first. The city council is concerned with optimizing the traffic flow and wants to calculate the time it would take an average citizen to travel from their home to their work place, given a particular configuration of the ILM traffic lights.



You have been contracted by the OTC to provide key simulation data that will help the Octavia City Council determine the number and placement of the ILM Traffic Lights. To do that, you will be given a series of test cases that include speed limit data, the proposed location of traffic lights, the initial reset time of each traffic light, a starting point and time, and a destination point for a driving test (both points are intersections). You are to determine the minimum time it would take an

Octavian driver to traverse the major streets and avenues from the starting point to the destination point with the given configuration. Remember that Octavians are very strict and law-abiding citizens – they always drive at exactly the speed limit and always obey traffic lights. Keep in mind that no left or right turns are permitted on a red light in the City of Octavia.

Input

There will be multiple test cases, sequentially numbered starting with 1. The input begins with a single line containing an integer giving the number of tests. Following, there will be three separate lines of input for each test case, with all data items separated by whitespace (spaces or tabs):

- Speed limits: a list of street or avenue names on which the speed limit is set to 20mph. All other streets and avenues (not listed in this input line) have a speed limit of 30mph.
- Intersections with traffic lights and their reset times: a list of intersection labels, each followed by a reset time in minutes. All the reset times are given as integral numbers of minutes after midnight.
- Start location, start time, end location: The intersection label of the start location, the time (as an integral number of minutes past midnight) when the Octavian driver leaves the start location, followed by the destination intersection label. Assume that if the start and destination locations have traffic lights, then the driver does not have to wait on those lights in order to depart or to arrive at the destination.

Output

For each test case, display the test case number and the number of minutes it takes the Octavian driver to reach the destination. Use a format similar to that shown in the samples. Display a blank line after the output for each case.

Sample Input

```
2
B C D 4
D3 0    B4 2    C4 0    E4 0    E5 4
E2 10   B5

A1 0    A2 0
A1 4    A2
```

Output for the Sample Input

```
Case 1: 14 minutes

Case 2: 2 minutes
```

Problem 4: Bridge Construction

In a remote area of Canada, a bridge between two islands is being repaired. During the repair process the bridge can only take traffic in one lane. The bridge runs east and west, and there is a flag person with a radio at each end of the bridge to control the traffic flow.

Cars may arrive at the east end of the bridge with their driver wishing to go west. Cars may also arrive at the west end of the bridge with their drivers heading east. This being Canada, all drivers obey the speed limit exactly – it takes precisely 60 seconds to cross the bridge in either direction.

Of course, when the traffic flow is from east to west, any cars arriving on the west end must wait; similarly, if the traffic is moving across the bridge from east to west, cars arriving at the east end must wait. The very instant that the last car going a certain direction (say, west to east) has crossed the bridge, the traffic can then run in the opposite direction (east to west).

The input data for this problem consists of a sequence of arrival events, each event identifying the time a car arrives and at which end of the bridge it arrives. Your task is to determine when the last of the cars to arrive has left the bridge.

Example

Each event is characterized by the time since the previous event, and the end of the bridge at which the car arrives. For example, suppose the following happens:

Time	End of the bridge
0	E
50	E
15	W

On the bridge, the following happens:

Absolute Time	Description
0	A car arrives at the east end. Since it is the first car it starts to cross, east to west.
50	A second car arrives; since the direction is currently east to west, it also is allowed to cross.
60	The first car leaves the bridge, and the second car is 10 seconds across.
65	A third car arrives at the west end of the bridge. It must wait; the direction right now is east to west. Note that this happens at time 65 because it is 15 seconds later than the previous event in the data.
110	The second car is now across. Since there are no more cars on the east end, the radios are used to instantly reverse the flow to west to east, and the third car starts across.
170	The third car takes 60 seconds to cross, and has reached the end of the bridge. As there are no waiting cars, and no further arrival events to consider, we display the result and proceed to the next set of data.

It is possible to have more than one car waiting at either end of the bridge. Since it is effectively a one lane bridge during the repair, cars can only proceed single file, cars travelling in the same direction across the bridge must maintain at least one second spacing. For example, consider this data:

Time	End of the bridge
0	E
10	W
10	W

On the bridge, the following happens:

Absolute Time	Description
0	A car arrives at the east end. Since it is the first car it starts to cross, east to west.
10	A second car arrives; because it is on the west end it must wait.
20	Another car has arrived at the west end of the bridge. It must wait as well.
60	The first car leaves the bridge; the west to east traffic can proceed, and the second car starts across.
61	One second later the other car on the west end begins to cross.
121	This last car takes 60 seconds to cross, and has reached the end of the bridge.

The program should display "121 seconds" and proceed to the next set of data.

Input

There will be multiple input cases, sequentially numbered starting with 1. The first line of the input contains a single integer specifying the number of cases. Following this, each case has one event per line specified by an integer ∆T to indicate the time since the previous event, whitespace, and either the character 'E' or the character 'W' to indicate at which end of the bridge the car arrived. The absolute time at which each data set begins is 0, the first event always has ∆T = 0, and ∆T for all other events is greater than 0. There is a line containing the integer –1 after the last line of data for each case.

Output

For each case, display the case number and the number of seconds required for all the cars to cross the bridge. If each allowable sequence of events does not yield the same time required for all cars to cross the bridge, then you should report the minimum of those times. Display the output in a format similar to that shown in the samples. Display a blank line after the output for each case.

Sample Input

```
3
0 E
50 E
15 W
-1
0 E
10 W
10 W
-1
0 W
30 E
10 W
10 W
40 W
-1
```

Output for the Sample Input

```
Case 1. 170 seconds

Case 2. 121 seconds

Case 3. 210 seconds
```

Problem 5: Clock Repair

Mr. Horologia's House of Clocks contains various cuckoo clocks that customers have brought in for repair. Since they are in the clock shop, one might rightly assume that these clocks don't quite run as they should. In fact, a fast clock may take 3,500 seconds to advance one hour, instead of 3,600 seconds. A slow clock might take 3,750 seconds.

Every midnight on Sunday morning, Mr. Horologia sets all clocks to exactly 12:00. He has sufficient assistants awake at that hour that all clocks can be set simultaneously. Some time later, possibly that same day but quite possibly several days later, all clocks in the room will cuckoo at precisely the same instant in time. (All clocks initially would chime when they are set at midnight on Sunday, but the initial cuckooing does not count.) What is the first day, hour, minute, and second when all clocks simultaneously go off? The time might be several days in the future; also, midnight on the next Sunday morning might come around again before they ever cuckoo simultaneously. In this latter case, Mr. Horologia will set them all again, and the correct answer would be "Never".

Example

An easy example involves two clocks, one that advances an hour every 3,000 seconds, and a second clock that advances an hour every 4,500 seconds. For simplicity, note that these represent 50 minutes and 75 minutes, respectively. The first clock would reach 1:00 AM after 50 minutes, then 2:00 AM after 100 minutes, and 3:00 AM after 150 minutes. The second clock would reach 1:00 at 75 minutes and 2:00 at 150 minutes. Thus, 150 minutes after midnight, clock number one and clock number two both cuckoo. The correct answer is thus "Sunday at 2:30:00 AM".

A second example might involve four clocks, all of which are extremely fast. They advance at 600, 1200, 1800, and 600 seconds. After 30 minutes, all of them cuckoo except for the second clock. After 60 minutes, all will cuckoo. Thus the answer here is "Sunday at 1:00:00 AM".

Finally, suppose we have four clocks with speeds of 3601, 3559, 3600, and 3700. The answer in this case is "Never" – the next Sunday will occur before all clocks cuckoo.

Input

There will be multiple cases, sequentially numbered starting with 1. The input for each case is a single line that contains integers giving the number of clocks *N*, followed by *N* "seconds" measurements. A line containing the integer 0 follows the last case. Because the repair shop is limited, *N* will never be larger than 10.

Output

For each case, display the case number and either the first date and time when all clocks will cuckoo simultaneously, or the word "Never" as described above. Display a blank line after the output for each case.

Sample Input

```
2 3000 4500
4 600 1200 1800 600
3 3550 3650 3655
2 3525 3625
0
```

Output for the Sample Input

```
Case 1: Sunday at 2:30:00 AM

Case 2: Sunday at 1:00:00 AM

Case 3: Never

Case 4: Friday at 9:58:45 PM
```

Problem 6: Arbitrary Multiplication

In traditional "long multiplication" we determine the product of two integers, X and Y, by multiplying X by the individual digits of Y, in turn, starting with the units digit. The results of these multiplications are arranged appropriately and added, yielding the completed product.

The representation of these operations is usually done in a particular manner. Consider the multiplication of 123 by 95:

```
  123
   95
  ---
  615
1107
-----
11685
```

The numbers to be multiplied, X and Y, are each displayed on a separate line, followed by a horizontal line. The results of multiplying X by each digit of Y are then displayed on separate lines, followed by another horizontal line, and then the final product. In this problem you are to perform a sequence of such multiplications, displaying the results in this traditional representation, with the horizontal lines displayed using hyphens (minus signs).

Input

There will be multiple input cases, sequentially numbered starting with 1. The input for each case will be a single line containing two integers, X and Y, separated by whitespace (one or more blanks and tab characters). Whitespace may also precede the first integer and follow the second integer. Each integer will have no more than 30 digits. A line containing only an end of line character follows the last case.

Output

For each input case, first display the case number on a line by itself. Then perform the multiplication of X by Y, displaying the results in the form shown above and in the examples shown below. Follow the output for each multiplication by a blank line. If Y contains only a single significant digit, omit the second horizontal line and the sum (since in that case it would be superfluous). Display zeroes only when they are significant. There must be no unnecessary leading blanks on any output line. That is, the entire multiplication display must appear as far to the left as possible.

The number of hyphens in the first horizontal line should be the same as the number of significant digits in the larger of X and Y. The number of hyphens in the second horizontal line, if it is produced, should be the same as the number of significant digits in the product of X and Y.

Sample Input

```
  4 7
135  46
    12345   862
```
this line contains only an end of line

Output for the Sample Input

```
Case 1:
 4
 7
 -
28

Case 2:
 135
  46
 ---
 810
540
----
6210

Case 3:
   12345
     862
   -----
   24690
  74070
 98760
--------
10641390
```

Problem 7: Genome Matching

Genes are DNA sequences that cause certain traits in living organisms. For example, a specific DNA sequence, called the sickle cell gene, manifests itself in Sickle Cell Anemia, a deadly disease of the blood. Similarly, other genes are responsible for traits like blue or brown eyes, blond or brunette hair, and so on[1]. The human DNA code, called the human genome, has already been extensively studied and fully mapped out to genes. Thus, it is possible to collect a DNA sample from a person and predict some of the traits in that individual, caused by genes found in the sample.

Your task is to locate and identify a number of pre-defined genes in several given DNA sequences. All DNA molecules, as well as any gene, are sequences composed from only four distinct chemicals, called nucleotides (Adenine, Cytosine, Guanine, and Thymine), which are labeled with the following letters: A, C, G, and T. From biology, we know a DNA molecule forms a right-handed double-helix structure where each base from one of the strands connects to exactly one corresponding base from the second strand. The bases always form A-T or C-G pairs (or, reflexively, T-A, and G-C) and thus, any gene sequence always has its complement in the pairing strand. Due to the method of collecting and examining DNA material, any given gene DNA sequence may be found in the DNA sample in either its original sequence, or in reverse, or in its complement image, or its reverse complement image. For example:

$$\text{The gene ACTTAGAAGGT may be found as } \begin{cases} \text{ACTTAGAAGGT (original)} \\ \text{TGAATCTTCCA (original's complement)} \\ \text{TGGAAGATTCA (original reversed)} \\ \text{ACCTTCTAAGT (reversed complement)} \end{cases}$$

Further, even though unlikely, some genes, or one of their alternative sequences, may be fragments of other, larger genes. Finally, the same gene may be found in the same DNA sample in multiple locations, either in its original, or one of its alternative sequences. You will be given a set of genes, one gene per input line, each labeled with a simple name. The name and the gene sequence will be separated by a colon (":"). For example:

```
Blue_eyes:ACTTAGAAGGT
Blond_hair:CTTAAGGGCGGGCTTCTTTA
...
```

The list of genes will be followed by one or more sample DNA sequences, which are submitted to you for your evaluation. Your program must display the list of genes present in each sample DNA sequence and the location (counting from 1) of the first base of the that gene, in the order in which those genes were encoded by the sequence. For example, the sequence

```
ACCTTACCCTTAAGGGCGGGCTTCTTTAGTGCTTGAGGACCTTCTAAGT
TTACGGAATAACGATTTCTTCGGGCGGGAATTCCAAGTCCCGGATCGA
```

encodes Blond_hair , Blue_eyes, and Blond_Hair at locations 9, 39, and 63, respectively. If the DNA sequence does not encode any of the known genes, simply output "none".

[1] *Admittedly, this is an oversimplification of the structure and function of genes, but it will do for this problem.*

Input

The first line of input will contain a single integer *NG* indicating the number of genes that are supplied. This number will be no larger than 20. *NG* lines will follow this line, each containing a simple name for a gene (1 to 18 characters), a colon, and the gene sequence (no more than 40 characters each of which is A, T, C, or G). There may be whitespace following the gene before the end of line.

Following the *NG* lines containing genes there will appear an arbitrary number of sample DNA sequences, none of which is longer than 5000 characters. These are numbered sequentially, starting with 1. Each sample sequence will begin on a new line, and will naturally be composed from the letters A, T, C, and G. There may be multiple lines required for a single sample sequence, and the last letter in the last line of the sample will be followed immediately by a period ("."). There may be whitespace at the end of each line (including the last in a sample). The sample input illustrates this format.

A line containing only a single period in the first column follows the last sample.

Output

For each sample DNA sequence, display the sequence number (1, 2, …). If none of the NG genes appear in the sequence, then display the word "none" on the same line as the sequence number. Otherwise, display the names of the genes – in the order they appear – on separate lines following the sequence number. After the name of each gene encountered in the DNA sequence, output the index of the first matching nucleotide of the gene (counting from 1). If multiple genes match at the same index, they should be displayed in the same order they appear in the input. Each gene will match the DNA sequence at most once for a given index value. Your output should be very similar to that shown below, including the indentation of the names of the genes. Display a blank line after the output for each sample DNA sequence.

Sample Input

```
2
Blue_eyes:ACTTAGAAGGT
Blond_hair:CTTAAGGGCGGGCTTCTTTA
ACCTTACCCTTAAGGGCGGGCTTCTTTA
GTGCTTGAGGACCTTCTAAGT
TTACGGAATAACGATTTCTTCGGGCGGGAAT
TCCAAGTCCCGGATCGA.
.
```

Output for the Sample Input

```
Sequence 1:
    Blond_hair at 9
    Blue_eyes at 39
    Blond_hair at 63

Sequence 2: none
```

Problem 8: Look, Ma! No Ruler!

In this problem you are given a piece of graph paper with one-inch squares. The paper is *W* inches wide and *H* inches tall, and you want to identify two points that are close as possible to *D* inches apart, where *D* is a real number. Naturally we can only accurately identify distances between the points formed by the intersections of the horizontal and vertical lines on the graph paper. Assume each point is identified by a pair of integers (*X* , *Y*), where *X* is the horizontal distance (in inches) to the right of the left edge of the paper and *Y* is the vertical distance down from the top edge of the paper. Thus a page that is 8 inches wide and 10 inches tall will have the point (0,0) in the upper left corner and the point (8,10) in the lower right corner. We assume that all distances will be measured from the upper left corner, (0,0). Your task is to find the point (*X* , *Y*) that is as close as possible to D inches from (0,0). If there are multiple points meeting this requirement, pick the point that has the smallest possible *X* value.

Input

There will be multiple cases to consider, numbered sequentially starting with 1. For each case there will be a single line of input containing *W*, *H*, and *D* separated by whitespace. Each of *W* and *H* will be no larger than 1000. A line containing three zeroes will follow the input for the last case.

Output

For each case display the case number and the integers *X* and *Y* in the exact format shown in the samples below. Display a blank line after the output for each case.

Sample Input

```
8 10 1.5
12 25 7.7
0 0 0
```

Output for the Sample Input

```
Case 1: (1,1)

Case 2: (3,7)
```

Problem 9: Hex Carries

Adding hexadecimal numbers is not too difficult, but you must remember the carries! Like adding decimal numbers, carries of 1 are generated when the sum of two digits and the carry from the right exceeds the maximum value that can be represented in a single digit (15, of course, for hexadecimal digits). In this problem you will be given a pair of hexadecimal numbers and asked to determine the number of non-zero carries that are involved in adding these numbers. We don't care about the sum.

For example, suppose the numbers to be added are 1A2B and D6E4. The following steps would be used to determine the number of carries.

- B (decimal 11) and 4 are added, yielding a sum of 15 (there is no carry into the low-order position). Since this is less than 16, no carry is generated.
- 2 and E (decimal 14) are added (with no carry from the preceding step), yielding a sum of 16. Since this is too large for a single hexadecimal digit (it's larger than 15), a carry is generated to the next step.
- A (decimal 10) and 6 are added, along with the carry from the previous step. The sum is 10 + 6 + 1 = 17, and a carry is generated.
- Finally, 1 and D (decimal 13) and the carry from the previous step are added yielding 1 + 13 + 1 = 15, and no carry is generated.

So a total of 2 carries were generated, and this is the answer for this case.

Input

There will be multiple cases to consider, each numbered sequentially starting with 1. For each case the input consists of a single line containing two non-zero hexadecimal numbers, each containing no more than 30 hexadecimal digits. The allowable hexadecimal digits are 0 through 9 and 'A' through 'F' and 'a' through 'f'. These numbers are separated, and possibly preceded and followed by spaces. The input for the last case is followed by a line containing two hexadecimal numbers, each consisting of a single '0'.

Output

For each case display the case number and the number of carries generated. Use a format **identical** to that shown in the samples below, and display a blank line after the output for each case.

Sample Input

```
1A2B D6E4
1 f
   1    2222222222
     0   0
```

Output for the Sample Input

```
Case 1: 2 carries

Case 2: 1 carry

Case 3: No carries
```