Problem 1: Soundex

A problem that has received considerable attention over the years is that of transforming a name into some code that tends to bring together all the variants of the same name. This is useful in applications involving spoken names when there is a good chance that the name will be misspelled. The following "soundex" algorithm, which was originally developed by Margaret K. Odell and Robert C. Russell, has often been used for encoding surnames. Assume the input is a single name consisting of only upper or lower case alphabetic characters with no punctuation, and that case is not significant in alphabetic comparisons.

1. Retain the first letter of the name (in upper case), and drop all occurrences of a, e, h, i, o ,u, w, and y in other positions of the name.
2. Assign the following numbers to the remaining letters after the first letter:

b, f, p, v → 1	l → 4
c, g, j, k, q, s, x, z → 2	m, n → 5
d, t → 3	r → 6

3. If two or more letters with the same numeric code (from step 2) were adjacent in the original name (before step 1), then omit all but the first of them.
4. Convert to the form "letter, digit, digit, digit" by adding trailing zeros if there are fewer than three digits, or by dropping the rightmost digits if there are more than three digits.

Examples

As our first example, suppose we want the code for the name "Euler". In step 1 we retain the first E, but drop the u and the second e, yielding "Elr". In step 2 we assign codes to l and r, yielding E46. Step 3 doesn't apply in this case. In step 4 we must add an additional 0 to obtain the desired letter, digit, digit, digit format, so our final code is E460.

Now consider generating the code for "Gauss". We keep the G and drop the a and the u, yielding "Gss". After step 2 we have G22. In step 3 we note that the two "2" codes were yielded by two occurrences of "s" in the original name, so we keep only one, yielding G2. Finally, we add two zeroes to give the final code, G200.

Input

There will be multiple names to convert using this soundex algorithm. Each input line contains a single name consisting only of upper and lower case alphabetic characters, starting in the first column, optionally followed by blanks, then the end of line. No name will have more than 20 alphabetic characters. The line containing the last name will be followed by a line containing a period in column one.

Output

For each input name, display a single output line containing the code (one uppercase alphabetic character and three digits), a tab character (or four spaces), and the input name.

Sample Input

```
Euler
Gauss
Hilbert
Knuth
Lloyd
.
```

Output for the Sample Input

```
E460    Euler
G200    Gauss
H416    Hilbert
K530    Knuth
L300    Lloyd
```

Problem 2: Election Results

The newly liberated country of Freedestan is preparing for its first democratic elections. Your company has won the bid for providing election automation to Freedestan and has been contracted by the Freedestani Election Council (FEC) to develop a ballot counting machine to be used in this election. Your software team is tasked with developing the Ballot Analysis and Result Recording (BARR) software module of the machine.

Your BARR module will be extensively tested by FEC representatives to ensure accuracy and compliance with the FEC rules and regulations. The tests will be performed using a series of ballots from several sample elections as input.

Unmarked Ballot Format

An unmarked ballot consists of one or more text lines, each of which contains the following:

OfficeName: Candidate*1* Candidate*2* ... Candidate*N*

This indicates that the individuals with names "Candidate*1*", "Candidate*2*", ... and "Candidate*N*" are running for office "OfficeName". The name of each office and the name of each candidate will contain at least two characters. Whitespace (blanks and/or tabs) will appear before each candidate's name and may optionally follow the last candidate's name. The names of candidates for an office may appear in different orders on different ballots, although the names of all candidates will be present. The lines for offices are presented in the same order on each ballot, which is in descending order of importance—the line for the most important office appears first and the line for the least important office appears last. Each office has at least one candidate.

Marking a Ballot

A voter properly selects a candidate for an office by placing a single "X" or "x" in the space immediately preceding the name of that candidate on the ballot so that some whitespace appears before and after the "X" or "x". For example, the line

President: X Amabo Gomaa

indicates a vote for the candidate "Amabo" for the office of "President". Unfortunately, it is also possible for a voter to place additional "X" or "x" marks on a line, and even other single-character marks. The ballot marking mechanism guarantees that there will always be some whitespace on each side of each voter-placed mark, and that marks must appear after the colon following the name of the office. Ballot lines containing more than one mark, or any mark other than "X" or "x" will cause the ballot containing that line to be flagged for examination by an election official. All ballot lines will be counted if they contain only a single "X" or "x" mark properly preceding the name of a candidate, even on ballots flagged for examination.

Tallying the Election Results

A candidate may run for one or more offices, but may only win election to one office. A candidate wins election to an office if he/she receives the most valid votes for that office, and has not been elected to or tied for a more important office. In case of a tie, all tied candidates for that office will be identified in the election results; each tied candidate is also ineligible for election to a less important office. It is possible that no candidates remain eligible for an office after applying these rules. In this case, the office is not filled by the election and the situation is reported by the BARR module software.

Input

To test your BARR module, the FEC will provide ballots for an arbitrary number of elections. The ballots for each election will be preceded by a line containing integers *NO* and *NB*. *NO* is the number of offices appearing on each ballot, and will be no larger than 10. *NB* is the number of ballots provided for the test election. Next will appear the *NO* × *NB* lines for the marked test ballots; the first *NO* lines contain the first ballot, the next *NO* lines contain the second ballot, and so forth. A line containing two zeroes will follow the last ballot for the last election. The FEC guarantees that there will be no more than 100 unique candidate names in any test election.

Output

For each test election, your BARR module should display the results of the election beginning with a line identifying the election sequence number (1, 2, ... corresponding to the input order). For each office, in descending order of importance, display the name of the office indented from the left column, a colon, the name of the candidate that won, or the names (in ascending alphanumeric order) of the candidates that tied for, election to that office, and the number of valid votes they received, separating each of these with a space. In case no valid candidates remain for an office, display "`--UNFILLED--`" instead of the winner's name and votes. Use one line per office. After the last office, on a separate line starting with "Invalid ballots: ", report in ascending order the sequence number of those ballots that contain one or more invalid votes - these ballots will have to be handled manually. Sequence numbers are not part of the input, but can be easily derived from the order in which the ballots appear in the input stream, counting from 1. If no ballots in an election were invalid, then omit the "Invalid ballots" line. Follow the entire report for an election with a single blank line. Use the format shown in the samples.

Sample Input

```
2 6
President:  X Buruni   Amabo  Gomaa
VicePresident:  x Buruni   Chakravarti
President:   Buruni  X Amabo  Gomaa
VicePresident:  Buruni   Chakravarti
President:   Buruni   X Amabo  Gomaa
VicePresident:  X Buruni  x Chakravarti
President:   Buruni x  Amabo  Gomaa
VicePresident:  Buruni   X Chakravarti
President:   @ Buruni   Amabo  Gomaa
VicePresident:  Buruni   x Chakravarti
President:   Buruni   Amabo  x Gomaa
VicePresident:  x  Buruni   Chakravarti
2 ~~8~~ 7
ChiefCook:  x Fred    Bob    Sue    Joe
BottleWasher: X Joe  Fred    Sue    Bob
ChiefCook:  x Fred    Bob    Sue    Joe
BottleWasher:  x Fred    Sue    Joe    Bob
ChiefCook:    Fred  x Bob    Sue    Joe
BottleWasher:  x Fred    Sue    Joe    Bob
ChiefCook:    Fred  x Bob    Sue    Joe
BottleWasher:  x Fred  x Sue  x Joe x  Bob
ChiefCook:    Fred    Bob    Sue    Joe    x
BottleWasher:  x Fred    Sue    Joe     Bob
ChiefCook:    Fred    Bob    Sue    Joe
BottleWasher:  x Fred    Sue    Joe    Bob
ChiefCook:    Fred    Bob    Sue  x Joe
BottleWasher:  x Fred    Sue  x Joe    Bob
0 0
```

Output for the Sample Input

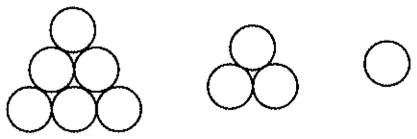
```
Election 1 results:
  President: Amabo 3
  VicePresident: Buruni Chakravarti 2
  Invalid ballots: 3, 5

Election 2 results:
  ChiefCook: Bob Fred 2
  BottleWasher: Joe 1
  Invalid ballots: 4, 5, 7
```

Problem 3: Cannonball Pyramids

If you visit historical battlegrounds from periods when cannon were used, you may see cannonballs (metal spheres) stacked in various ways. In this problem we're interested in the number of cannonballs in such pyramids with triangular bases — that is, with cannonballs arranged in an equilateral triangle on the base, then similar equilateral triangles of cannonballs stacked on top of that, until a single cannonball is placed on the top. This three-sided pyramid is, in fact, one of the Platonic solids, a tetrahedron.

For example, suppose we begin with a base containing 6 cannonballs arranged in an equilateral triangle as shown in the figure on the left; this equilateral triangle has 3 cannonballs on each edge. The next layer will contain 3 cannonballs, 2 on each edge — as shown in the middle. The final layer always contains just one cannonball. The total number of cannonballs in this pyramid is thus 6 + 3 + 1 = 10. In general, each layer of such a pyramid will have one fewer cannonballs on each edge of the triangle than the next lower layer.



The number of cannonballs in a layer is just the sum of the integers from 1 to *N*, where *N* is the number of cannonballs on one side of the layer.

Given the number of cannonballs on each side of the base, compute the total number of cannonballs in the entire tetrahedral pyramid.

Input

The first line of the input contains a single integer *N* giving the number of cases posed, for a maximum of 100 cases. Following that are exactly N lines, each with a single integer giving the number of cannonballs on each side of the base for a tetrahedron of cannonballs, a number less than 1000.

Output

For each input case, display the case number (1, 2, ...), a colon and a blank, the number of cannonballs on each side of the base, one blank, and finally the total number of cannonballs in the tetrahedron.

Sample Input	Output for the Sample Input
5 3 5 27 999 1	1: 3 10 2: 5 35 3: 27 3654 4: 999 166666500 5: 1 1

Problem 4: Pencils from the Nineteenth Century

The program "Sunday Week-end Edition" on the US National Public Radio (NPR) network has a "Sunday Puzzle" segment. The show that aired on Sunday, June 29, 2008 presented the following puzzle:

> From a nineteenth century trade card advertising Bassett's Horehound Troches, a remedy for coughs and colds: A man buys 20 pencils for 20 cents and gets three kinds of pencils in return. Some of the pencils cost 4 cents each, some are two for a penny and the rest are four for a penny. How many pencils of each type does the man get?

A clarification provided to the problem indicated that correct solutions would contain at least one of each pencil type.

This is an enhancement of the problem that originally aired on the NPR show. Rather than just considering 20 pencils for 20 cents, consider the case of *N* pencils for *N* cents. Given a value of N, determine if a solution is possible, and if so, determine all possible solutions.

Input

Each input line, except the last, contains a value of *N* in the range 1 to 256 for which the problem is to be solved. The last input line contains the integer 0.

Output

For each value of *N* in the input, display the case number (1, 2, ...) and the phrase "*N* pencils for *N* cents" as shown in the sample output below. If there are no solutions for a particular value of N, then display the line "No solution found." If there are solutions, display three lines for each one, separating the groups of three lines for each solution by a blank line. Order these solutions by increasing numbers of four-cent pencils. Display a blank line after the output for each case.

Sample Input

```
10
20
40
0
```

Output for the Sample Input

```
Case 1: 10 pencils for 10 cents
No solution found.

Case 2: 20 pencils for 20 cents
3 at four cents each
15 at two for a penny
2 at four for a penny

Case 3: 40 pencils for 40 cents
6 at four cents each
30 at two for a penny
4 at four for a penny

7 at four cents each
15 at two for a penny
18 at four for a penny

```

Problem 5: Fibonacci Period

Almost everyone is familiar with the Fibonacci sequence which begins 1, 1, 2, 3, 5, ... Each term in this sequence (except for the first two) is the sum of the two preceding terms, so the terms continually become larger and never repeat.

But what can we say about a parallel sequence in which each term is equal to the corresponding term in the Fibonacci sequence, taken modulo N, for $N > 1$? That is, suppose we label the terms in the Fibonacci sequence $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, $F_4 = 3$, $F_5 = 5$, ... Then label the terms in the parallel sequence $P_i = F_i \bmod N$.

For example, suppose N = 4. We have the following correspondence between the sequences:

F =	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	...
P =	1	1	2	3	1	0	1	1	2	3	1	0	1	1	2	...

Note that the sequence P repeats after the first six elements. That is, the sequence P has a period of 6.

Given a value of N greater than 1 and not greater than 1,000,000,000, what is the period of the corresponding parallel sequence?

Input

There will be multiple values of N to consider. Each value will appear on a line by itself. The last value of N will be followed by a line containing an integer less than or equal to 1.

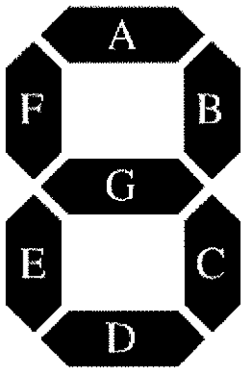
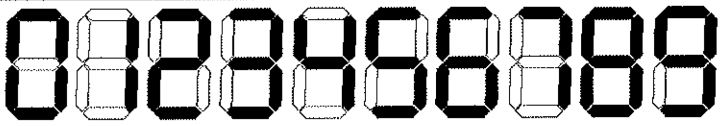
Output

For each input case, display the case number (1, 2, ...), a colon and a blank, the value of N, a blank, and the period of the parallel sequence.

Sample Input	Output for the Sample Input
2	1: 2 3
3	2: 3 8
4	3: 4 6
5	4: 5 20
0	

Problem 6: Painted Calculator

Patrick has a four function calculator; unfortunately his sister has decided to be a traditional "evil sibling" and has painted over the display. The calculator still functions, but of course you can't see the results. This calculator has a display constructed from seven-segment LEDs. The figure on the left below illustrates how the seven LED segments at each digit position are arranged. The figure on the right shows which segments are illuminated to display each of the ten decimal digits.

Seven-Segment display	The decimal numbers
	

This particular calculator only displays three digits of accuracy. There are three seven segment digits, and there are no decimal points since all math is performed using integers (with truncation where appropriate).

If the number on the display happens to be negative, there is a minus sign in front of the number. For a number such as -6, this is done by making the rightmost digit a "6" and using the middle segment of the next digit over as the minus sign. For a number such as -123, there is one additional segment to the left of the display. Thus, the number -112 lights up 10 segments, made up of 9 for the number, and one more for the minus sign. The display thus ranges from "-999" to "999". There is no "plus sign" for positive numbers.

In order to prove that the calculator still operates, and justify scraping off the paint, the back cover is removed and an ammeter is attached to the display. An ammeter measures the current consumed by the device, and each segment of a digit consumes 5 milliamps. Thus, to display a number such as "798" the current consumption can be predicted as follows:

Digit 7 - 3 segments lit = 15 milliamps
Digit 9 - 6 segments lit = 30 milliamps
Digit 8 - 7 segments lit = 35 milliamps
Total = 80 milliamps

Of course, it's apparent that the number "897" also consumes the same amount of current, as does "789" or, for that matter, "-891".

The calculator allows you to enter numbers with an optional minus sign and up to three digits, and to either add, subtract, multiply, or divide the numbers. Patrick wishes to prove that the functionality of the calculator is intact, so he enters "949" and measures 80 milliamps. Then he pushes the subtract key. Next he enters "51" and measures 35 milliamps. After pressing the "=" key the result should be "898", measuring 100 milliamps. As a second example, he enters "-5", then pushes the addition key. He then enters "-4" and presses equal. The answer is "-9"; the measurements were 30 milliamps, 25 milliamps, and 35 milliamps respectively.

Patrick's sister claims that this is a trick, so she presents him with the following problem: given the current consumption (in milliamps) of operand X, operand Y, and result Z, and given an unknown operation Op, determine the number of possible values for X Op Y = Z, assuming that possible solutions exist. If no such solutions exist, the answer is "No solutions."

Note that in all cases, there are only three digits on the display. Although the input values of 80, 35, and 100 potentially could represent 12337=949*13, this is not possible because 12337 is too large for the three digit display to hold.

Example

An example is as shown above; for inputs 80, 35, and 100 one possibility is "925 - 117 = 808". Some input combinations may have no result. For the inputs 35, 10, and 10 the answer is "No solutions." For any given input set, the program should either print "No solutions." or it should print the number of solutions found.

Input

There will be multiple cases. Each input line contains values for X, Y, and Z. Each number is in milliamps according to the above description. A line containing a single zero follows the input data for the last case.

Output

For each case, display "Case " and the case number (1, 2, ...), a colon and a blank, the number of solutions (or "No" if there are no solutions), and then the word "solution" or "solutions" (as appropriate) and a period. Your output should exactly follow this format, as is illustrated by the sample input and output shown below.

Sample Input

```
30 10 10
35 10 10
15 20 30
30 65 65
0
```

Output for the Sample Input

```
Case 1: 1 solution.
Case 2: No solutions.
Case 3: 9 solutions.
Case 4: 819 solutions.
```

Problem 7: 3-Trees

We are all familiar with the traditional binary search tree (BST) in all of its variants. There are AVL trees, red-black trees, etc. In each, the search strategy is the same – if the key of the item for which we are searching is less than the key in the tree node we are examining, then the search should proceed down the left sub-tree, if it exists. Likewise, if the key is greater than that in this tree node, search the right sub-tree. Suppose we call the key associated with a node V, and the key for which we are searching S. Then if $S < V$ go to the left, and if $S > V$ go to the right. If $S = V$ we have found that node for which we were searching.

A 3-tree is either empty, or consists of a node and three sub-trees, each of which is a 3-tree. If a 3-tree is non-empty, and the key in the node is V, then

- if a node has a key less than V/2, then that node is in the "far left" sub-tree of the node containing V.
- if a node has a key greater than or equal to V/2 and less than V, then that node is in the "near left" sub-tree of the node containing V.
- if a node has a key greater than V, then that node is in the "right" sub-tree of the node containing V.
- if a node contains exactly the key V which is being sought, then none of the sub-trees need be considered in a 3-tree search.

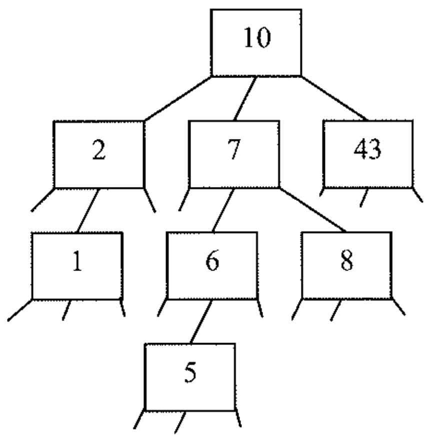
Obviously this may skew the tree, but this is not an issue for this problem. All comparisons should be done using integer arithmetic and truncation, so that a node with a value 9 would have a "far left" sub-tree that contains only nodes with keys less than 4, and a "near left" sub-tree that contains nodes with keys greater than or equal to 4 and strictly less than 9.

Assume the level *L* of a tree node is zero for the root of the tree, one for nodes immediately below the root, and so on. Your task in this problem is to read a set of numbers and construct a 3-tree using the numbers as keys in the order given. Then, after the tree is constructed, display the following three values:

- the maximum value of *L* when visiting only "far left" links starting from the root,
- the maximum value of *L* when visiting only "near left" links starting from the root,
- the maximum value of *L* when visiting only "right" links starting from the root.

Example

An example is shown here. The input keys are 10, 2, 43, 7, 1, 8, 6, 5, which creates the 3-tree shown at right. The level for the tree node containing 2 is 1, the level for the node containing 5 is 3, and the level for the node containing the 43 is 1. The values to be displayed are therefore 1, 3, and 1.



Input

There will be multiple cases to consider. Each input case consists of a set of one or more integers greater than zero followed by a zero. The integers in each of these sets – with the exception of the zero that marks the end of the set – are to be used, in order, to construct a 3-tree for the current input case. The set of integers for the last case is followed by a single zero - that is, a set that would yield an empty 3-tree. None of the integers is larger than 10,000.

Values may appear more than once in an input set, but only one 3-tree node is to be created for each unique input value.

Output

For each input case, display "Case " followed by the input case number (1, 2, ...), a colon and a blank, and three numbers separated by a blank. These three numbers are the largest possible values for L along the appropriate "far left", "near left" and "right" paths in the 3-tree constructed from the input data for the current case.

Sample Input

```
10
2
43
7
1
8
6
5
0
220
631
571
332
104
502
567
854
0
0
```

Output for the Sample Input

```
Case 1: 1 3 1
Case 2: 1 0 2
```

Problem 8: Egyptian Multiplication

Ancient Egyptian multiplication is a systematic method for multiplying two numbers that does not require the multiplication table, but only the ability to multiply by 2, and to add. Also known as Egyptian multiplication and Peasant multiplication, it decomposes one of the multiplicands into a sum of powers of two and creates a table of doublings of the second multiplicand. This method may be called mediation and duplication, where mediation means halving one number and duplication means doubling the other number.

This method has three phases: the decomposition, the table and the result.

The decomposition of a number N thus consists of finding the powers of two which make it up. The Egyptians knew empirically that a given power of two would only appear once in a number. For the decomposition, they proceeded methodically; they would initially find the largest power of two less than or equal to the number in question, subtract it out and repeat until nothing remained. (The Egyptians did not make use of the number zero in mathematics).

Example

First consider the decomposition of the number N = 13:

- the largest power of two less than or equal to 13 is 8, 13 - 8 = 5;
- the largest power of two less than or equal to 5 is 4, 5 - 4 = 1;
- the largest power of two less than or equal to 1 is 1, 1 - 1 = 0.

N = 13 is thus the sum of these powers of two: 8, 4 and 1.

After the decomposition of the first multiplicand (N), it is necessary to construct a table of powers of two times the second multiplicand (M) from one up to the largest power of two found during the decomposition. In the table, a line is obtained by multiplying the preceding line by two.

For example, if the largest power of two found during the decomposition of N = 13 is 8 and M = 238, the table is created as follows:

Power of 2	M × Power of 2
1	238
2	476
4	952
8	1904

Finally, the result is obtained by adding the numbers from the second column for which the corresponding power of two makes up part of the decomposition of M. In this case, the powers of two involved are 1, 4, and 8. Thus, the result of the multiplication of 13 x 238 is obtained as the 1904 + 952 + 238 = 3094.

note: clarification on swap of M + N was made!

Input

There will be multiple pairs of multiplicands *N* and *M* provided as input. Each multiplicand will be between 1 and 10,000. Each of the input lines, except the last, will contain a pair of values for *N* and *M*. The last line of input will contain the single integer -1.

Output

For each pair of multiplicands, display the case number (1, 2, ...), followed by the multiplicands (*N* and *M*) separated by " x ", an equal sign, and an expression giving the sum of the multiples of M that are to be added to yield the product of M and N, in order from smallest to largest. Use the format shown in the sample output below.

Sample Input

```
13 238
1000 1
1 1
-1
```

Output for the Sample Input

```
Case 1: 13 x 238 = 238 + 952 + 1904
Case 2: 1000 x 1 = 8 + 32 + 64 + 128 + 256 + 512
Case 3: 1 x 1 = 1
```

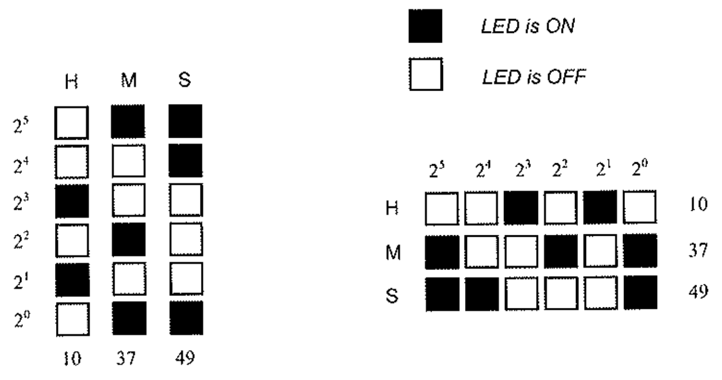
Problem 9: Binary Clock

A binary clock is a clock which displays traditional sexagesimal time (military format) in a binary format. The most common binary clock uses three columns or three rows of LEDs to represent zeros and ones. Each column (or row) represents a single decimal digit in a format known as binary-coded decimal (BCD).

When you uses three columns (vertically), the bottom row in each column represents 1 (or 2^0), with each row above representing higher powers of two, up to 2^5 (or 32). To read each individual digit in the time, the user adds the values represented by each illuminated LED, and then reads these from left to right. The first column represents the hour, the next column represents the minute, and the last column represents the second.

When you uses three rows (horizontally), the right column in each row represents 1 (or 2^0), with each column left representing higher powers of two, up to 2^5 (or 32). To read each individual digit in the time, the user adds the values represented by each illuminated LED, and then reads these from above to below. The top row represents the hour, the next row represents the minute, and the bottom row represents the second.

Example



The time shown in this example is 10:37:49.

The Problem

In this problem you are provided with a textual representation of the reading of a binary clock in row-major order. That is, each row of LEDs, whether the clock is vertical or horizontal, is represented as a sequence of text strings, with "()" used to represent an LED that is off, and "(*)" used to represent an LED that is on. For example, the reading from the vertical clock (shown on the left above) would be ()(*)(*)()()(*)(*)()()()(*)()(*)()()()(*)(*), and the reading from the horizontal clock would be ()()(*)()(*)()(*)()()(*)()(*)(*)(*)()()()(*). Given an indication of whether the clock was vertical or horizontal, your task is to convert this textual representation to the traditional time representation (10:37:49 in this case).

Input

There will be multiple cases in the input. The first line of input contains an integer *N* that specifies the number of cases. Each of the remaining *N* lines contains the following items, in the order specified:

- optional whitespace (blanks and/or tab characters)
- a string with a textual representation of the binary clock (no embedded whitespace)
- at least one whitespace character
- a lowercase letter "h" or "v" indicating if the clock was horizontal or vertical
- optional whitespace
- end of line

Output

For each input case, display a line that first contains the case number and a colon. If the syntax of the textual representation of the time is incorrect (that is, it does not contain 18 combinations of "()" and "(*)"), display the phrase "incorrect syntax". If the syntax is correct but the time represented is incorrect (that is, hours is greater than 23, or minutes or seconds is greater than 59), display the phrase "incorrect time". Otherwise display the time in the traditional format. Display two digits for hours that are greater than 9, otherwise display only a single hours digit. Always display two digits for minutes and seconds.

Sample Input

```
4
() () (*) () (*) () (*) () () (*) () (*) (*) (*) () () () (*)  h
() (*) (*) () () (*) (*) () () () (*) () (*) () () () (*) (*)  v
() (*) (*) () () (*) () () () () () (*) () () () () () (*)  h
() (*) () () () (*) () () () () () (*) () () () () (*) () () () ()  h
```

Output for the Sample Input

```
Case 1: 10:37:49
Case 2: 10:37:49
Case 3: incorrect time
Case 4: incorrect syntax
```