Coding Challenge

	Problem	**Color**
1	Excel-lent!	Orange
2	Sinus Rhythm	Purple
3	Old School	Red
4	Tipping Nest	Silver
5	YOLO - TTL	Yellow

Problem 1

Excel-lent!

A certain spreadsheet program, whose name shall not be uttered here, labels the columns of a spreadsheet using letters. Column 1 is labeled as "A", column 2 as "B", ..., column 26 as "Z". When the number of columns is greater than 26, another letter is used. For example, column 27 is "AA", column 28 is "AB" and column 52 is "AZ". It follows that column 53 would be "BA" and so on. Similarly, when column "ZZ" is reached, the next column would be "AAA", then "AAB" and so on.

- The rows in the spreadsheet are labeled using the row number. Rows start at 1.
- The designation for a particular cell within the spreadsheet is created by combining the column label with the row label. For example, the upper-left most cell would be "A1". The cell at column 55 row 23 would be "BC23".

You will write a program that converts numeric row and column values into the appropriate cell address for the heretofore-as-yet-unnamed spreadsheet application.

Input

Input consists of lines of the form: RnCm. n represents the row number $[1, 300000000]$ and m represents the column number, $1 \leq m \leq 300000000$. The values n and m define a single cell on the spreadsheet. Input terminates with the line: `R0C0` (that is, n and m are 0). There will be no leading zeroes or extra spaces in the input.

Output

For each line of input (except the terminating line), you will print out the appropriate cell address for the specified cell as described above.

Sample Input

```
R1C1
R3C1
R1C3
R299999999C26
R52C52
R53C17576
R53C17602
R0C0
```

Sample Output

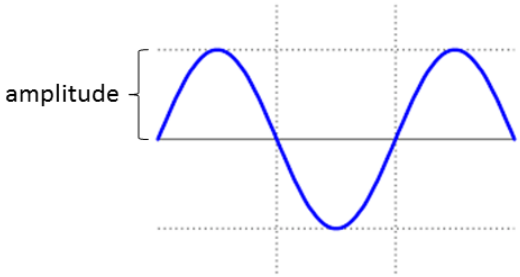
```
A1
A3
C1
Z299999999
AZ52
YYZ53
YZZ53
```

Problem 2

Sinus Rhythm

As described in the Wiki-est o'Pedias (and illustrated below), a *sinusoid* is

> "...a mathematical curve that describes a smooth repetitive oscillation. It is named after the function sine, of which it is the graph. It occurs often in pure and applied mathematics, as well as physics, engineering, signal processing and many other fields."



You will write a program that simulates sinusoids—*ascii art style*. (Yeah!) To simulate plotting sinusoids, write a program that receives amplitude and string pairs and plots them on the console.

Input

The input consists of one or more pairs containing an amplitude and a string of characters. The amplitude appears first on its own line, followed by a line containing a string of characters.

Output

The output will be an ascii art graph containing all of the sinusoids plotted together, in the order that they appear in the input. (Thus, if the graphs overlap, the character of a later string will overwrite those of earlier strings in the position where they overlap.

As shown in the example below, a string of amplitude 0 is not plotted, and a string of amplitude ± 1 is plotted as a straight line.

The overall graph will be as wide as the longest string, and will have a height equal to twice the largest amplitude, minus 1.

Sample Input

```
1
--------------------------------
6
++++++++++++++++++++++++++++++++
0
XXXXXXXXXXXXX
-3
@@@@@@@@@@@@@@@@@@@@@@
```

Sample Output

```
     +                    +
    + +                  + +
   +   +                +   +
  +   @ +     @        +     +
 +   @ @ +   @ @      +       +
@---@---@-+-@---@---@----------+--
 @ @     @ @     @ @
  @       @ +     @
             +   +
              + +
               +
```

Problem 3

Kickin' it Old School

Before cell phones were smart phones (i.e. when they were still dumb phones), texting was done rather tediously using the number pad. The numbers 2 through 9 each represented three or four characters. If a letter was the ith character on a number, you had to multi-tap that number i times to get that letter. You want an "s" on a word? Press the 7 key four times. The letters associated with each number are 2: ABC, 3: DEF, 4: GHI, 5: JKL, 6: MNO, 7: PQRS, 8: TUV, and 9: WXYZ.

Today, most phones that only have numeric key pads use predictive algorithms so users do not have to multi-tap keys. Before you can develop predictive texting, you need to know all possible strings associated with a sequence of numbers.

Write a program that, given a dictionary of words, and a sequence of numbers from 2 through 9, outputs all possible dictionary words represented by that sequence.

Input

The first line will be an integer $1 \leq n \leq 100000$ representing the length of the dictionary to follow. (An example dictionary of words might be found in `/usr/share/dict/words`).

The next n lines will contain strings representing entries in the dictionary.

The remaining strings, one per line, will be integer sequences typed in using the phone's "keyboard".

Output

For each string, output all of the words from the dictionary that could match. The sequences will not be multi-tap sequences. If there are no dictionary matches, output all possible strings represented by that sequence. Outputs for each sequence should be sorted alphabetically.

Examples

```
Input:
1
QQ
27

Output:
AP
AQ
AR
AS
BP
BQ
BR
BS
CP
CQ
CR
CS

Input:
6
JOVE
CLOVE
BAD
LOUD
LOVE
GLOVE
5683
223

Output:
JOVE
LOUD
LOVE
BAD
```

Problem 4

Tipping the Nest

Markup languages such as HTML use tags to highlight sections with special significance. In this way, a sentence in boldface can be indicated thus:

> <B>**This is a sentence in bold.**</B>

Many so-called "paired" tags have an opening tag of the form `<TAG>` and a closing tag of the form `</TAG>`, such that portions of text can be bracketed as above. Tags can then be combined to achieve more than one effect on a particular piece of text simply by nesting them properly, for instance:

> <I><B>***This text is italic and bold.***</B></I>

You are writing a grading system for CPSC 110. Two of the most common mistakes when students first start writing HTML are:

- getting the nesting wrong:
  <B><DIV>These tags are wrongly nested.</B></DIV>
- forgetting a tag:
  <body>This should be bold, but there is a missing closing body tag.

Write a program to check that all the tags in a given segment of HTML code are correctly nested, and that there are no missing or extra tags. An opening tag is enclosed by angle brackets, and contains a string representing the tag name, followed by zero or more attributes. White space may appear around tag names and attributes. HTML tag names are **not** case-sensitive—**you should convert all tag names to lowercase when you read them in**. All of the following are valid opening tags:

```
<a href="http://cnu.edu">
< boDy >
<select  id='dropdown' size= 1 >
```

For paired tags, a corresponding closing tag will be the same string used to open the tag, preceded by the symbol "/"—but not necessarily in the same case.

```
</A >
< /BODY>
</ select>
```

Some tags are unpaired tags. For this problem, *assume all tags are paired except for* `input, doctype, br, img, hr, basefont, meta, and link`.

Input

The input will consist of any number of segments of HTML to be validated. Segments may consist of multiple lines, terminating with a `#` which will not occur anywhere else in the text. The input will never break a tag between two lines. The input will be terminated by an empty paragraph, e.g., a line containing only a single "`#`".

Output

If the HTML segment is correctly tagged then output the line "`VALID`", otherwise output a line of the form "`INVALID: Expected <expected>, found <unexpected>`" where `<expected>` is the closing tag matching the most recent unmatched tag, and `<unexpected>` is the actual closing tag encountered. If either of these is the end of the HTML segment, i.e. there is either an unmatched opening tag or no matching closing tag at the end of the paragraph, then replace the tag or closing tag with the symbol "`#`". These points are illustrated in the examples below which should be followed exactly as far as spacing is concerned. Singleton (unpaired) tags listed above should be ignored.

Sample Input

```
The following text<C><B>is centered and in boldface</B></C>#
<B>This <\g>is <B>boldface</B> in <<*> a</B> <\6> <<d>sentence#
<B><C> This should be centered and in boldface, but the
tags are wrongly nested </B></C>#
<B>This should be in boldface, but there is an extra closing
<img src='tag.gif'>tag</B></C>#
<B><C>This should be centered and in boldface, but there is
a missing closing tag</C>#
#
```

Sample Output

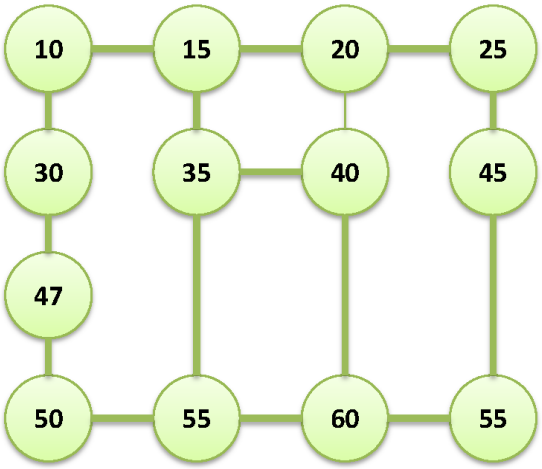
```
VALID
VALID
Expected </c>, found </b>
Expected #, found </c>
Expected </b>, found #
```

Problem 5

YOLO (TTL)

To avoid the potential problem of network messages (packets) looping around forever inside a network, each message includes a Time-To-Live (TTL) field. This field contains the number of nodes (stations, computers, routers, etc.) that can re-transmit a message, forwarding it along toward its destination, before the message is unceremoniously dropped. Each time a node receives a message it decrements the TTL field by 1 before forwarding it on. If the destination of the message is the current station, then the TTL field's value is ignored. However, if the message must be forwarded, and the decremented TTL field contains zero, then the message is not forwarded.

In this problem you are given the description of a number of networks, and for each network you are asked to determine the number of nodes that are **not** reachable given an initial node and TTL field value. Consider the following example network:



If a message with a TTL field of 2 was sent from node 35 it could reach nodes 15, 10, 55, 50, 40, 20 and 60. It could not reach nodes 30, 47, 25, 45 or 65, since the TTL field would have been set to zero on arrival of the message at nodes 10, 20, 50 and 60. If we increase the TTL field's initial value to 3, starting from node 35 a message could reach all except node 45.

Input

There will be multiple network configurations provided in the input. Each network description starts with an integer NC specifying the number of connections between network nodes. An NC value of zero marks the end of the input data. Following NC there will be NC pairs of positive integers. These pairs identify the nodes that are connected by a communication line. There will be no more than one (direct) communication line between any pair of nodes, and no network will contain more than 500 nodes. Following each network configuration there will be multiple queries as to how many nodes are **not** reachable given an *initial node* and *TTL setting*. These queries are given as a pair of integers, the first identifying the starting node and the second giving the initial TTL field setting. The queries are terminated by a pair of zeroes.

Output

For each query, display a single line showing the test case number (numbered sequentially from one), the number of nodes not reachable, the starting node number, and the initial TTL field setting. The sample input and output shown below illustrate the input and output format.

Sample Input

```
16
10 15   15 20   20 25   10 30   30 47   47 50   25 45   45 65
15 35   35 55   20 40   50 55   35 40   55 60   40 60   60 65
35  2   35  3    0  0

14
1 2   2 7   1 3   3 4   3 5   5 10   5 11
4 6   7 6   7 8   7 9   8 9   8  6   6 11
1 1   1 2   3 2   3 3   0 0

0
```

Sample Output

```
Case 1: 5 node(s) unreachable from node 35 with TTL = 2.
Case 2: 1 node(s) unreachable from node 35 with TTL = 3.
Case 3: 8 node(s) unreachable from node 1 with TTL = 1.
Case 4: 5 node(s) unreachable from node 1 with TTL = 2.
Case 5: 3 node(s) unreachable from node 3 with TTL = 2.
Case 6: 1 node(s) unreachable from node 3 with TTL = 3.
```