Problem 3—GIF Decompression

Professor Plum enjoys telling stories to students about his adventures in industry before he joined the faculty. In 1987 he was working at CompuServe when the Graphics Interchange Format (GIF) encoding was created to compress image files. This was an inspiring event is his life, but lately he has been having trouble remembering how the decompression algorithm works and has asked you to help him.

In this problem, we will consider a simplified version of the GIF encoding algorithm applied to strings of alphabetic characters. Essential for this compression is a dictionary that assigns numeric encodings (we will use base 10 numbers for this problem) to different strings of characters. The dictionary is initialized with mappings for characters or substrings which may appear in the string. For example, if we expect to encounter all 26 letters of the alphabet, the dictionary will initially store the encodings (`A,00`), (`B,01`), (`C,02`), …, (`Z,25`). If we are compressing DNA data, the dictionary will initially store only 4 entries: (`A,0`), (`T,1`), (`G,2`) and (`C,3`). Note that the length of each initial encoding is the same for all entries (2 digits in the first example, and 1 digit in the second).

The compression algorithm proceeds as follows:

1. Find the longest prefix of the uncompressed portion of the string which is in the dictionary, and replace it with its numeric encoding.
2. If the end of the string has not been reached, add a new mapping **(*s*,*n*)** to the dictionary, where ***s*** = the prefix just compressed plus the next character after it in the string, and ***n*** = the smallest number not yet used in the dictionary.

For example, assume we started with the string `ABABBAABB` and a dictionary with just two entries, (`A,0`) and (`B,1`). The table below shows the steps in compressing the string.

String	Longest Prefix	Replaced With	New Dictionary Entry
ABABBAABB	A	0	(AB,2)
0BABBAABB	B	1	(BA,3)
01ABBAABB	AB	2	(ABB,4)
012BAABB	BA	3	(BAA,5)
0123ABB	ABB	4	-

The final compressed string is `01234`.

There is only one other rule: the replacement strings used are always the size of the longest encoding in the dictionary at the time the replacement occurs. Thus, with the dictionary above, if the string to compress is long enough that an entry of the form (*s*, `10`) is added to the dictionary, then from this point on all numerical replacement strings used in the compressed string must be expanded to 2 digits long (i.e., `A` will now be encoded as `00`, `B` as `01`, `AB` as `02`, etc.); if an entry (*s*', `100`) is added to the dictionary, all replacements from this point forward will increase to 3 digits long, and so on. Thus, the longer string `ABABBAABBAABAABAB` will be encoded as `01234027301`, not `0123402731`. Try it!

OK, now that you are experts at compressing, it's time to relax and decompress!

INPUT SPECIFICATION

Each test case will consist of two lines. The first line will contain a string of digits to decompress. The second line will contain the initial dictionary used in the compression. This line will start with a positive integer ***n*** indicating the number of entries in the dictionary ($1 \le n \le 100$), followed by ***n*** alphabetic strings. The first of these will be paired with 0 in the dictionary (or 00 if $n > 10$), the second with 1, and so on. The last test case will be followed by a line containing a single 0.

OUTPUT SPECIFICATION

For each test case, output a single line containing the word `Case`, a single space, the case number (starting at 1), a colon, a single space, and the decompressed string. All input strings will have been legally compressed.

SAMPLE INPUT

```
01234
2 A B
01234027301
2 A B
02151120182729
26 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
21104
3 BA A C
01
2 JA VA
0
```

SAMPLE OUTPUT

```
Case 1: ABABBAABB
Case 2: ABABBAABBAABAABAB
Case 3: CPLUSPLUS
Case 4: CAABAAA
Case 5: JAVA
```