Data compression is easy if you know some theory behind, like Markov models and such. However, although the theory is kind of complex, compressing data is not so difficult if you take the direct results from the theory.

For example, compressing means "predicting" the next character after a set of characters given as input, based on the current history of the document. If you predict the character correctly, you don't have to output anything, as it follows the "pattern" or "rules" of the text.

Suppose we have an input document composed of n characters, c_0 to c_{n-1}. Suppose also that we are processing the document at character i, that is, c_i. We maintain a 2D-matrix, P with the predictions for a given pair of characters. Then, the compression is simple: if $P(c_{i-2}, c_{i-1}) = c_i$ then, don't output anything (the actual character has been predicted); if not, output the character c_i literally, and update P so that the next time $P(c_{i-2}, c_{i-1}) = c_i$.

Surprisingly, this simple mechanism gives fair compression ratios. However, you have to be careful with the codification of the compressed text, because you have to be able to tell whether a character has been predicted or not.

For that purpose, the output is divided in groups, G_0 to G_m, with $m = \lceil \frac{n}{6} \rceil$. Each group G_k contains six elements (bytes) and a descriptor byte:

$$G_k = \langle b_k, g_0^k, g_1^k, g_2^k, g_3^k, g_4^k, g_5^k \rangle$$

Thus, for the first group, g_0^0 refers to c_0 , g_1^0 to c_1 and so on. For the G_k group, g_0^k refers to c_{6*k}, g_1^k refers to c_{6*k+1}, and so on. The value of each g_j^k is as follows:

$$g_j^k = \begin{cases} \lambda & \text{if} \quad P(c_{6*k+j-2}, c_{6*k+j-1}) = c_{6*k+j} \\ c_{6*k+j} & \text{if} \quad P(c_{6*k+j-2}, c_{6*k+j-1}) \neq c_{6*k+j} \end{cases}$$

Note that λ means that no character is produced, so that the actual size of the group can be less than 7 bytes (producing compression).

Note also that g_j^k are only valid for those k and j so that $6 * k + j < n$. This means the last group can be incomplete to suit real file length.

b_k is defined as follows:

$$b_k = 64 + \sum_{i | g_i^k = \lambda} 2^i$$

Suppose also that $c_{-1} = 0$ and $c_{-2} = 0$.

The purpose of this problem is to build a compressor following this specification.

Input

The input is formed by a series of n bytes (characters) finalized by an EOF. That is, you have to take all the input as the text to be compressed (including spaces and carriage return).

Output

The output is the set of $m = \lceil \frac{n}{6} \rceil$ groups $(G_0 \dots G_{m-1})$ in form of ASCII characters, as described previously.

Sample explanations: In spite of the description above, consider the Sample Input below as two different files (one per non-empty line) just to illustrate the way the compressor works. Then the results at the Sample Output depends only of the corresponding line.

Case 1: In the second group (the one starting with "`q`") the first "`o`" is predicted because in the first "`lo`", space after "`l`" was followed by an "`o`". Also, the 5th and 6th characters of this group are predicted. Therefore, the descriptor byte is: "`q`" = ASCII(113) = 1110001b.

The output is 23 bytes long, and the input 24 bytes long, so in this short text, the algorithm compressed 1 bytes out of 24 (4.16% compression).

Case 2: In this case, 10 bytes are compressed (20.4% compression).

Sample Input

```
a lo loco lo coloco lola

football is football and basketball is basketball
```

Sample Output

```
@a lo lqco @ colocI lla

@footba@ll is |foGand@ baskettblVibs_lA
```