acm International Collegiate Programming Contest
Sponsored by IBM®

25th Annual ACM International Collegiate Programming Contest
2000 Greater New York Regional

Greater NY Region

Problem F – Entropy

Background

An entropy encoder is a data encoding method that achieves lossless data compression by encoding a message with “wasted” or “extra” information removed. In other words, entropy encoding removes information that was not necessary in the first place to accurately encode the message. A high degree of entropy implies a message with a great deal of wasted information; english text encoded in ASCII is an example of a message type that has very high entropy. Already compressed messages, such as JPEG graphics or ZIP archives, have very little entropy and do not benefit from further attempts at entropy encoding.

English text encoded in ASCII has a high degree of entropy because all characters are encoded using the same number of bits, eight. It is a known fact that the letters E, L, N, R, S and T occur at a considerably higher frequency than do most other letters in english text. If a way could be found to encode just these letters with four bits, then the new encoding would be smaller, would contain all the original information, and would have less entropy. ASCII uses a fixed number of bits for a reason, however: it’s easy, since one is always dealing with a fixed number of bits to represent each possible glyph or character. How would an encoding scheme that used four bits for the above letters be able to distinguish between the four-bit codes and eight-bit codes? This seemingly difficult problem is solved using what is known as a “prefix-free variable-length” encoding.

In such an encoding, any number of bits can be used to represent any glyph, and glyphs not present in the message are simply not encoded. However, in order to be able to recover the information, no bit pattern that encodes a glyph is allowed to be the prefix of any other encoding bit pattern. This allows the encoded bitstream to be read bit by bit, and whenever a set of bits is encountered that represents a glyph, that glyph can be decoded. If the prefix-free constraint was not enforced, then such a decoding would be impossible.

Consider the text “AAAAABCD”. Using ASCII, encoding this would require 64 bits. If, instead, we encode “A” with the bit pattern “00”, “B” with “01”, “C” with “10”, and “D” with “11” then we can encode this text in only 16 bits; the resulting bit pattern would be “0000000000011011”. This is still a fixed-length encoding, however; we’re using two bits per glyph instead of eight. Since the glyph “A” occurs with greater frequency, could we do better by encoding it with fewer bits? In fact we can, but in order to maintain a prefix-free encoding, some of the other bit patterns will become longer than two bits. An optimal encoding is to encode “A” with “0”, “B” with “10”, “C” with “110”, and “D” with “111”. (This is clearly not the *only* optimal encoding, as it is obvious that the encodings for B, C and D could be interchanged freely for any given encoding without increasing the size of the final encoded message.) Using this encoding, the message encodes in only 13 bits to “0000010110111”, a compression ratio of 4.9 to 1 (that is, each bit in the final encoded message represents as much information as did 4.9 bits in the original encoding). Read through this bit pattern from left to right and you’ll see that the prefix-free encoding makes it simple to decode this into the original text even though the codes have varying bit lengths.

As a second example, consider the text “THE CAT IN THE HAT”. In this text, the letter “T” and the space character both occur with the highest frequency, so they will clearly have the shortest encoding bit patterns in an optimal encoding. The letters “C”, “I’ and “N” only occur once, however, so they will have the longest codes. There are many possible sets of prefix-free variable-length bit patterns that would yield the optimal encoding, that is, that would allow the text to be encoded in the fewest number of bits. One such optimal encoding is to encode spaces with “00”, “A” with “100”, “C” with “1110”, “E” with “1111”, “H” with “110”, “I” with “1010”, “N” with “1011” and “T” with “01”. The optimal encoding therefore requires only 51 bits compared to the 144 that would be necessary to encode the message with 8-bit ASCII encoding, a compression ratio of 2.8 to 1.

Input

The input file will contain a list of text strings, one per line. The text strings will consist only of uppercase alphanumeric characters and underscores (which are used in place of spaces). The end of the input will be signalled by a line containing only the word "END" as the text string. This line should not be processed.

Output

For each text string in the input, output the length in bits of the 8-bit ASCII encoding, the length in bits of an optimal prefix-free variable-length encoding, and the compression ratio accurate to one decimal point.

Example

Input	Output
AAAAABCD THE_CAT_IN_THE_HAT END	64 13 4.9 144 51 2.8