Compact Encoding

Time limit: 2 seconds
Memory limit: 1024 megabytes

Binary formats often use compact representations for integers. Consider writing a 32-bit unsigned integer to a file: you must always reserve 4 bytes to represent it (remember, 8 bits make one byte). However, in many real-life applications, integer values tend to be small. Writing these small values using a fixed 4-byte representation results in files that are mostly filled with zero bytes.

To make the representation more compact, we introduce the following encoding scheme. Each value is represented as a sequence of bytes $b_1, b_2, \ldots, b_k$, where each b_i is an integer between 0 and 255, inclusive. The most significant bit of each byte serves as a continuation flag, and the lower 7 bits carry the actual data. If the continuation flag is 1, more bytes follow; for the last byte, it is 0. The representation is *big-endian*, meaning that b_1 contains the most significant bits of the encoded value.

For example, here's how to find the compact representation of $n = 112025$. First, we find its binary representation:

$$112025 = 11011010110011001_2$$

Next, we split it into 7-bit chunks, padding with zeros on the left if necessary:

$$0000110 \ / \ 1101011 \ / \ 0011001$$

The first two chunks have a following chunk, so the corresponding bytes have their most significant bit set to 1. The last chunk has no following chunk, so its most significant bit is 0. This gives us:

$$b_1 = 10000110_2 = 134$$
$$b_2 = 11101011_2 = 235$$
$$b_3 = 00011001_2 = 25$$

You are given an integer n, and your task is to find its compact representation.

Input

The only line contains a single integer n ($0 \le n \le 2^{31} - 1$).

Output

Print a sequence of integers between 0 and 255, inclusive, representing the compact encoding of n. The encoding must not contain leading bytes with zero data bits: that is, it may not start with 128.

Examples

standard input	standard output
112025	134 235 25
128	129 0
0	0
42	42
16384	129 128 0
2147483647	135 255 255 255 127