Zeros

80 points

Memory Limit: 2 MB

Time Limit: 2 seconds

The factorial of a non-negative integer *n*, denoted *n*!, is the product of all positive integers less than or equal to *n*. For example,

4! = 4 × 3 × 2 × 1 = 24.

By the convention of an empty product, 0! is defined to be 1.

Trailing zeros of a number are a sequence of 0s after which no other digits follow. For example, the number 6000 has 3 trailing zeros and the number 900010 has 1 trailing zero, whereas the integer 250342 has no trailing zeros.

Factorials of some numbers end in certain number of trailing zeros. For example,

5! = 5 × 4 × 3 × 2 × 1 = 120 and 120 has 1 trailing zero, and
12! = 479,001,600 has 2 trailing zeros.

Write a program to find all the integers in a given inclusive interval between two positive integers *a* and *b* whose factorials have exactly *c* number of trailing zeros.

INPUT

The first line of the input consists of two positive integers *a* and *b* separated by a white space that defines an inclusive interval [*a* .. *b*].
It is guaranteed that $1 \le a \le b \le 2{,}000{,}000$.
The second line of the input is a single integer $1 \le c \le 100{,}000$ representing the number of trailing zeros required in the factorials of the integers in the interval to be in the output.

OUTPUT

Output consists of a single line, which may contain either a single integer 0 or a sequence of integers each separated by a white space. If none of the factorials of the integers in the specified input interval end in exactly *c* number of trailing zeros, the only output should be 0. If there are one or more integers in the input interval whose factorials end exactly in *c* number of trailing zeros, the output should contain all these integers in increasing order of their values, each separated by a white space.

Sample Inputs/Outputs

Input	Output
7 17 1	7 8 9
4 4 100	0
1 2000000 95554	382235 382236 382237 382238 382239

Be warned that the factorial grows very fast, in fact, faster than exponential functions. Naively computing the factorials of large integers would certainly cause overflows on standard built-in data types and would definitely exceed the time limit specified for this problem.