Description

This problem is about implementing an algorithm for calculating $a^n \,(mod\ 2019)$, following the idea of the fast algorithm for computing powers of numbers (Binary Exponentiation).

Fast power, is a trick to efficiently calculate a^n. A direct computation by multiplying a with itself does not work well when n is too large. We know that $a^{b+c} = a^b \times a^c, a^{2b} = (a^b)^2$. The idea of binary exponentiation is utilize this fact to divide the computation into smaller tasks according to the binary representation of n.

First, we take the binary form of n. For example:

$3^{13} = 3^{(1101)_2} = 3^8 \times 3^4 \times 3^1$

Because n has $\lfloor log_2 n \rfloor + 1$ bits, we can calculate the values of $a^1, a^2, a^4, a^8, \cdots, a^{2^{\lfloor log_2 n \rfloor}}$ by multiplying only $\lfloor log_2 n \rfloor$ times. Take 3 as an example:

$3^1 = 3$

$3^2 = (3^1)^2 = 3^2 = 9$

$3^4 = (3^2)^2 = 9^2 = 81$

$3^8 = (3^4)^2 = 81^2 = 6561$

In order to calculate 3^{13}, we only need to find out which digits are 1 in the binary form of n, and take the multiplication of the corresponding values in the above sequence:

$3^{13} = 3^{(1101)_2} = 3^8 \times 3^4 \times 3^1 = 6561 \times 81 \times 3 = 1594323$

Formally, if we write n as binary $(n_t n_{t-1} \cdots n_1 n_0)_2$, , we have

$n = n_t 2^t + n_{t-1} 2^{t-1} + n_{t-2} 2^{t-2} + \cdots + n_1 2^1 + n_0 2^0$

where $n_i \in 0, 1$. Then

$a^n = \left(a^{n_t 2^t + \cdots + n_0 2^0}\right) = a^{n_0 2^0} \times a^{n_1 2^1} \times \cdots \times a^{n_t 2^t}$

Hint: To avoid overflow in multiplication, you will need to use the fact:
$(a \times b)\ mod\ c = (a\ mod\ c) \times (b\ mod\ c)\ mod\ c$

Input Format

Two integers in one line: a, n, where $1 \le a \le 100, 1 \le n \le 2 \times 10^9$

Output Format

One integer $a^n \,(mod\ 2019)$

Sample Input 1

2 3

Sample Output 1

8

Sample Input 2

3 13

Sample Output 2

1332