1 Problem

The Fibonacci sequence is one of the most famous sequences of integers, defined by a recursive function $F(n) = F(n-1) + F(n-2)$ where $F(0) = 0$ and $F(1) = 1$.
Inspired by the Fibonacci sequence, Bessie decides to create her own sequences. She finds two integers A, B and defines a sequence $F(n) = A \cdot F(n-1) + B \cdot F(n-2)$.
As usual, $F(0) = 0$ and $F(1) = 1$. Using this notation, the Fibonacci sequence can then be thought of as $A = 1$ and $B = 1$. However, Bessie doesn't know how to calculate terms quickly. Help her by finding the Nth term of the sequence (modulo $10^9 + 7$).

INPUT FORMAT:
The first line contains A, B, and N where $0 \leq N \leq 10^{18}$.

OUTPUT FORMAT:
A single integer $F(N)$ where F is defined above.

SAMPLE INPUT:
3 2 10

SAMPLE OUTPUT:
79647

2 Solution

Simply simulating the sequence by iteratively generating the next element would be $O(N)$, which is too slow. Instead, use fast matrix multiplication.

The common trick for the Fibonacci sequence is to let

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

M represents

$$\begin{bmatrix} F_{n-1} & 1 \\ F_{n-2} & 0 \end{bmatrix}$$

By an inductive argument, the upper left term of M^{n-1} gives the Nth term in the sequence. When computing the jth term of the sequence, compute $M \times M_{j-1}$.

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} & 1 \\ F_{n-2} & 0 \end{bmatrix} = \begin{bmatrix} F_{n-1} + F_{n-2} & 1 \\ F_{n-1} & 1 \end{bmatrix}$$

The left column of the matrix "propagates" the changes, updating the next term and copying the previous newest term to the older term.

In order to extend this reasoning to this problem, redefine M in terms of A and B:

$$M = \begin{bmatrix} A & B \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} A & B \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} & 1 \\ F_{n-2} & 0 \end{bmatrix} = \begin{bmatrix} AF_{n-1} + BF_{n-2} & 1 \\ F_{n-1} & 1 \end{bmatrix}$$

which is exactly the behavior we want.

In order to compute M^N quickly, use the standard fast exponentiation algorithm (rewrite the exponent in binary and successively square M).

```python
# modulo
M = 1000000007

def dotp(u: list, v: list) -> float:
    """ dot product """
    return sum(u[i]*v[i] for i in range(len(u)))

def col(m: list, i: int) -> list:
    """ column of a matrix """
    return [m[j][i] for j in range(len(m))]

def mod_mat_mult(a: list, b: list, m: int) -> list:
    """ matrix multiplication with modulo """
    return [[dotp(a[i], col(b, j)) % m for j in range(len(b[0]))] \
           for i in range(len(a))]

def identity(n: int) -> list:
    """ identity matrix of size nxn """
    return [[int(i == j) for j in range(n)] for i in range(n)]

def mod_mat_exp(A: list, k: int, m: int) -> list:
    """ returns A^k % m"""
    v = identity(len(A))
    while k > 0:
        if k & 1 == 1:
            v = mod_mat_mult(v, A, m)
        k >>= 1
        A = mod_mat_mult(A, A, m)
    return v

def mod_seq(a: int, b: int, n: int, m: int) -> int:
    """ returns the nth sequence element mod m"""
    A = [[a, b],
         [1, 0]
        ]
    return mod_mat_exp(A, n - 1, m)[0][0] if n > 0 else 0

A, B, N = map(int, input().split())
print(mod_seq(A, B, N, M))
```