Aritmética modular

Hasta ahora hemos visto como se pueden implementar las operaciones aritméticas para enteros representados en forma decimal. Python3 utiliza internamente una representación en base 2^{64} y llama a la librería GMP para hacer las operaciones aritméticas, que tiene implementados unos algoritmos algo mas eficientes que los que hemos estudiado. La siguiente tabla muestra las complejidades binarias (operaciones entre dígitos) de cada una de las implementaciones en función de la cantidad de dígitos totales de ambos operandos:

Operación en $\mathbb{Z}$	Algoritmos de la escuela	Clases anteriores	Python3/GMP
Suma, resta y comparación	$O(n)$	$O(n)$	$O(n)$
Multiplicación	$O(n^2)$	$O(n^{\log_2 3})$	$O(n \log(n) \log\log(n))$
División con resto	$O(n^2)$	$O(M(n))$	$O(M(n))$

Otro algoritmo importante que vamos a utilizar es el que calcula el divisor común mayor g entre dos enteros x, y dados y además obtiene a, b tales que $g = ax + by$. En el problema 3.7 vimos que eso puede hacerse en $O(n^2)$ operaciones binarias, donde n es la cantidad total de dígitos de x e y. El algoritmo de Euclides tiene la misma complejidad binaria $O(n^2)$.

Sea $N \geq 1$ un entero. Decimos que $a \equiv b \pmod{N}$ si y solo si $N \mid a-b$, es decir, si a y b tienen el mismo resto al dividirlos por N. Se comprueba inmediatamente que la relación de congruencia módulo N es reflexiva, simétrica y transitiva. Eso separa a los enteros en clases disjuntas. Por ejemplo, si $N = 2$, todos los números pares están en una misma clase y todos los impares en otra. En álgebra, se suele representar $[a]_N$ a la clase de equivalencia módulo N que contiene a a, es decir,

$$[a]_N = \{b \in \mathbb{Z} \,:\, b \equiv a \pmod{N}\}.$$

Cuando no haya ambigüedad con el valor de N, se puede simplemente indicar $[a]$, o incluso, $\overline{a}$.

La relación de congruencia módulo N satisface

$$a \equiv b \ \wedge c \equiv d \implies a + c \equiv b + d,\ a - c \equiv b - d,\ ac \equiv bd \pmod{N}$$

por lo que el conjunto de clases de equivalencia forma un anillo (con la suma y producto usual) que se denota $\mathbb{Z}/N\mathbb{Z}$.

Los elementos del anillo $\mathbb{Z}/N\mathbb{Z}$ pueden *representarse* unívocamente con los enteros entre 0 y $N-1$.

$$\mathbb{Z}/N\mathbb{Z} = \big\{[0], [1], [2], \ldots, [N-1]\big\}$$

De aquí en adelante, utilizaremos esa representación. La operación `a % N` de Python3 permite reducir un entero `a` cualquiera a su correspondiente representante en el rango $[0, N-1]$. En álgebra, es común escribir esa operación como $a \bmod N$.

Si $n = size(N) \approx \log_{10}(N)$, entonces las sumas y restas en $\mathbb{Z}/N\mathbb{Z}$ pueden hacerse siempre con $O(n)$ operaciones binarias. Por ejemplo, una suma en $\mathbb{Z}/N\mathbb{Z}$ requiere de una suma en $\mathbb{Z}$ de dos números de n dígitos, luego una comparación en $\mathbb{Z}$ con N para saber si el resultado se salió del rango $[0, N-1]$ y, en ese caso, una resta en $\mathbb{Z}$ con N. En el peor caso son tres operaciones en $\mathbb{Z}$ de números de a lo sumo $n+1$ dígitos, lo que tiene complejidad binaria $O(n)$. De forma similar se puede ver que la resta en $\mathbb{Z}/N\mathbb{Z}$ tiene también complejidad binaria $O(n)$.

La multiplicación en $\mathbb{Z}/N\mathbb{Z}$ se reduce a una multiplicación de enteros de n dígitos, cuyo resultado tendrá $2n$ dígitos, seguida de una división con resto por N. Esto tiene complejidad binaria $O(M(n))$, dependiendo de la implementación elegida para la multiplicación de enteros, y suponiendo que se use el método de Netwon para la división.

```python
def sumar_mod(a, b, N):           # 0 <= a,b < N, N >= 1
    c = a + b
    if c >= N:
        c -= N
    return c

def restar_mod(a, b, N):          # 0 <= a,b < N, N >= 1
    c = a - b
    if c < 0:
        c += N
    return c

def multiplicar_mod(a, b, N):     # 0 <= a,b < N, N >= 1
   c = a * b
   c %= N
   return c
```

Programa 1: `modular.py` (lineas 1–16)

Una de las operaciones mas importantes que vamos a necesitar es la potenciación. Una forma simple de calcular a^k para $a \in \mathbb{Z}/N\mathbb{Z}$ y $k \geq 0$ es mediante k multiplicaciones. Un bucle `for i in range(k):` es suficiente, pero lamentablemente eso hace que la complejidad binaria del método sea *exponencial* en el tamaño de k. Una mejor idea es utilizar el mismo truco que hicimos para el divisor común mayor, es decir, si k es par, calculamos primero $a^{k/2}$ y luego elevamos al cuadrado (una sola multiplicación), y si k es impar, hacemos $a^k = a \cdot a^{k-1}$ para reducirnos al caso par. De ese modo, solamente haremos $O(\log k)$ multiplicaciones en $\mathbb{Z}/N\mathbb{Z}$, lo que da una complejidad total $O(M(n) \log k)$.

```python
def potencia_mod(a, k, N):                # 0 <= a < N, k >= 0, N >= 2
    if k == 0:                            # caso base (k = 0)
        r = 1                             # convencion: 0^0 = 1
    elif k % 2 == 0:                      # k es par (k > 0)
        r = potencia_mod(a, k//2, N)
        r = multiplicar_mod(r, r, N)
    else:                                 # k es impar (k > 0)
        r = potencia_mod(a, k-1, N)
        r = multiplicar_mod(a, r, N)
    return r
```

Programa 2: `modular.py` (lineas 18–27)

Problema 4.1. Demostrar que la sucesión de Fibonacci satistace

$$\begin{bmatrix} f_n \\ f_{n+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

y utilizar esa identidad para implementar un algoritmo que calcule eficientemente $f_k \bmod N$ dados $k \geq 0$ y $N \geq 1$. ¿Cuál es la complejidad binaria del algoritmo?

Sean $N_1, N_2 \geq 1$ tales que $N_1 \mid N_2$. Esto implica que si $a \equiv b \pmod{N_2}$ entonces también $a \equiv b \pmod{N_1}$, o en términos mas algebraicos, que la aplicación

$$\begin{aligned} \mathbb{Z}/N_2\mathbb{Z} &\to \mathbb{Z}/N_1\mathbb{Z} \\ [a]_{N_2} &\mapsto [a]_{N_1} \end{aligned}$$

está bien definida y es un morfismo de anillos. Por ejemplo, conocer la clase de un entero módulo 10, permite facilmente determinar la clase de equivalencia módulos 2 (la paridad) y módulo 5.

Teorema 4.1 (Teorema chino del resto)**.** *Sean $N, M \geq 1$ coprimos, es decir,* $\gcd(N, M) = 1$*. Entonces la aplicación*

$$\gamma : \mathbb{Z}/NM\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z} \oplus \mathbb{Z}/M\mathbb{Z}$$
$$[a]_{NM} \mapsto ([a]_N, [a]_M)$$

es un isomorfismo de anillos. En particular, para un $N = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ con $p_1, \ldots, p_k$ primos distintos y $n_1, \ldots, n_k \geq 1$ se tiene que $\mathbb{Z}/N\mathbb{Z} \simeq \mathbb{Z}/p_1^{n_1}\mathbb{Z} \oplus \cdots \oplus \mathbb{Z}/p_k^{n_k}\mathbb{Z}$.

Proof. Ya hemos visto que γ está bien definido y es un morfismo de anillos. Para comprobar que γ es un monomorfismo, basta ver que el núcleo de γ es trivial. Para eso, tomemos un elemento $[x]_{NM} \in \mathbb{Z}/NM\mathbb{Z}$ tal que $\gamma([x]_{NM}) = ([0]_N, [0]_M)$. Eso significa que tenemos un entero x que es divisible por N y por M, y por lo tanto, también por NM, ya que N y M no tienen factores en común. Eso prueba que $[x]_{NM} = [0]_{NM}$ en $\mathbb{Z}/NM\mathbb{Z}$ como necesitabamos probar. La suryectividad sigue de que ambos conjuntos tienen la misma cantidad de elementos y que la aplicación γ es inyectiva. □

La demostración del teorema anterior es *intencionalmente* no constructiva, con el objetivo de mostrar un problema muy frecuente el álgebra computacional. Hay una cantidad enorme de resultados de existencia y unicidad en álgebra (biyecciones entre objetos) cuya demostración no da un procedimiento explícito para calcular los objetos implicados.

Es claro que la aplicación $\gamma : \mathbb{Z}/NM\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z} \oplus \mathbb{Z}/M\mathbb{Z}$ del teorema 4.1 se puede implementar usando la operación % de Python3:

```
1 def gamma(x, N, M):
2     return (x%N, x%M)
```

Sin embargo no queda nada claro como implementar la inversa $\gamma^{-1} : \mathbb{Z}/N\mathbb{Z} \oplus \mathbb{Z}/M\mathbb{Z} \to \mathbb{Z}/NM\mathbb{Z}$. Si se lee cuidadosamente la demostración, se puede ver que la existencia y buena definición de γ^{-1} se sigue de la inyectividad de γ y que el dominio y codominio de γ tienen la misma cantidad de elementos. Es decir, $\gamma^{-1}(y)$ existe y está bien definido porque $\gamma(x)$ recorre todos los posibles elementos de $\mathbb{Z}/N\mathbb{Z} \oplus \mathbb{Z}/M\mathbb{Z}$ sin repetir cuando x recorre todos los elementos de con $\mathbb{Z}/NM\mathbb{Z}$. Eso es el equivalente a hacer una búsqueda exhaustiva.

```
1 def gamma_inversa(y, N, M):
2     for x in range(N*M):
3         if gamma(x, N, M) == y:
4             return x
```

Por supuesto, tal argumento es perfectamente válido para probar el teorema 4.1, pero no es de esperarse que el método sea el más eficiente.

Supongamos que $[x_1]_{NM} = \gamma^{-1}([1]_N, [0]_M)$ y $[x_2]_{NM} = \gamma^{-1}([0]_N, [1]_M)$, es decir,

$$\begin{array}{llll} x_1 \equiv 1 & (\text{mod } N) & x_1 \equiv 0 & (\text{mod } M) \\ x_2 \equiv 0 & (\text{mod } N) & x_2 \equiv 1 & (\text{mod } M) \end{array} \tag{1}$$

para ciertos enteros $0 \leq x_1, x_2 < NM$. Entonces, se tiene que $\gamma^{-1}([y_1]_N, [y_2]_M) = [x_1 y_1 + x_2 y_2]_{NM}$.

El problema se reduce a entender como determinar x_1. Como $x_1 \equiv 0 \pmod{M}$, buscamos un múltiplo de M, es decir, $x_1 = Mz_1$ para cierto $0 \leq z_1 < N$. La otra ecuación nos dice que $Mz_1 \equiv 1 \pmod{N}$. Si aplicamos el algoritmo del divisor común mayor extendido a N y M obtendríamos que $1 = \gcd(N, M) = aN + bM$ para ciertos $a, b \in \mathbb{Z}$. Esa identidad nos dice que $bM \equiv 1 \pmod{N}$ y por lo tanto podríamos tomar $z_1 = b \bmod N$ como la solución de la ecuación que nos falta.

```
1 def gamma_inversa_eficiente(y, N, M):
2     g, a, b = gcd_extendido_binario(N, M)  # obtenemos la tupla (g,a,b)
3     x1 = M * (b % N)                       # x1=1 (mod N), x1=0 (mod M)
4     x2 = N * (a % M)                       # x2=0 (mod N), x2=1 (mod M)
5     x = (x1*y[0] + x2*y[1]) % (N*M)        # combinacion lineal
6     return x
```

La hipótesis $\gcd(N, M) = 1$ implica que en la linea 2 obtendremos $g = 1$ y que $1 = aN + bM$ tal como necesitamos.

Problema 4.2. Sean $N_1, \ldots, N_k \geq 1$ coprimos y sea $N = \prod_{i=1}^{k} N_i$. Demostrar que el sistema de congruencias simultáneas $x \equiv a_i \pmod{N_i}$ para $a_1, \ldots, a_k$ dados es equivalente a una única congruencia $x \equiv a \pmod{N}$ para algún a. Implementar este resultado en Python3 como la función `reducir_congruencias(l)` que toma una lista `l = [(a1,N1),(a2,N2),...,(ak,Nk)]` de pares ordenados (tuplas) y devuelve el par `(a,N)`.

Problema 4.3. Generalizar el resultado del problema 4.2 al caso en que $N_1, \ldots, N_k$ no sean necesariamente coprimos. Más concretamente, probar que dados $a_1, \ldots, a_k \in \mathbb{Z}$, el sistema de congruencias simultáneas $x \equiv a_i \pmod{N_i}$ tiene solución si y solo si $a_i \equiv a_j \pmod{\gcd(N_i, N_j)}$ para todo i, j. En caso de tener solución, esta es equivalente a una única congruencia de la forma $x \equiv a \pmod{N}$, donde $N = \operatorname{lcm}(N_1, N_2, \ldots, N_k)$.

En la implementación de `gamma_inversa_eficiente(x,y,N)` hemos usado dos trucos importantes que se repiten muchas veces en álgebra computacional. El primero es la interpolación: basta con resolver las ecuaciones (1) y luego hacer combinaciones lineales de esas soluciones. El otro truco es obtener inversas modulares utilizando que la identidad de Bezout $gcd(x, y) = ax + by$ puede obtenerse algoritmicamente de forma eficiente. Si x e y son coprimos, entonces a es el inverso multiplicativo de x en $\mathbb{Z}/y\mathbb{Z}$ y b es el inverso multiplicativo de y en $\mathbb{Z}/x\mathbb{Z}$, es decir, $[a]_y[x]_y = [1]_y$ y $[b]_x[y]_x = [1]_x$.

Los elementos $[x]_N \in \mathbb{Z}/N\mathbb{Z}$ que tienen inversa multiplicativa corresponden unívocamente con los enteros $1 \leq x \leq N - 1$ coprimos con N. En el párrafo de arriba probamos que esa condición es suficiente. La necesidad es inmediata, ya que si $\gcd(x, N) = g > 1$, entonces $ax \bmod N$ será múltiplo de g para cualquier a y por lo tanto distinto de 1. Los elementos invertibles de $\mathbb{Z}/N\mathbb{Z}$ forman un grupo (con respecto a la multiplicación) que se denota $(\mathbb{Z}/N\mathbb{Z})^*$ o también $\mathcal{U}(\mathbb{Z}/N\mathbb{Z})$ llamado el grupo de unidades de $\mathbb{Z}/N\mathbb{Z}$. La cantidad de elementos de este grupo se denota $\varphi(N)$.

$$\varphi(N) = |(\mathbb{Z}/N\mathbb{Z})^*| = |\{x \,:\, 1 \leq x < N,\ \gcd(x, N) = 1\}|$$

Por ejemplo, si N es primo tenemos que $\varphi(N) = N - 1$, ya que todos los enteros entre 1 y $N - 1$ son coprimos con N. Esto nos dice que $\mathbb{Z}/N\mathbb{Z}$ es un cuerpo, ya que es un anillo (conmutativo) en el que todo elemento no nulo es invertible.

Todavía no tenemos la suficiente teoría para ver como es la estructura de $(\mathbb{Z}/N\mathbb{Z})^*$ en general, pero sí podemos ver algunos resultados básicos. Supongamos que $N = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ con $p_1, \ldots, p_k$ primos distintos y $n_1, \ldots, n_k \geq 1$. Por el teorema 4.1 tenemos que

$$(\mathbb{Z}/N\mathbb{Z})^* \simeq (\mathbb{Z}/p_1^{n_1}\mathbb{Z})^* \oplus \cdots \oplus (\mathbb{Z}/p_k^{n_k}\mathbb{Z})^*$$

y en particular

$$\varphi(N) = \varphi(p_1^{n_1}) \cdots \varphi(p_k^{n_k})$$

por lo que bastará estudiar el caso $N = p^n$ para cierto primo $p \geq 2$ y $n \geq 1$.

Claramente se tiene $\varphi(p^n) = p^n - p^{n-1}$, ya que hay exactamente p^{n-1} enteros $1 \leq x < p^n$ divisibles por p. Entonces:

$$\varphi(N) = \varphi\left(p_1^{n_1} \cdots p_k^{n_k}\right) = N\left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_k}\right).$$

Para terminar vamos a ver un resultado clásico de teoría de números.

Teorema 4.2 (Euler-Fermat)**.** *Sea $N \geq 1$ y sea a coprimo con N. Entonces $a^{\varphi(N)} \equiv 1 \pmod{N}$. En particular, si N es primo y $N \nmid a$, entonces $a^{N-1} \equiv 1 \pmod{N}$.*

Proof. Sea $S = \langle [a] \rangle$ el subgrupo de $(\mathbb{Z}/N\mathbb{Z})^*$ generado por $[a]$. Claramente se tiene que $S = \{[1], [a], [a^2], \ldots, [a^{s-1}]\}$ donde $s = |S|$ es el menor exponente tal que $[a^s] = [1]$. Por el teorema de Lagrange, sabemos que $|S| \mid |(\mathbb{Z}/N\mathbb{Z})^*|$, es decir, $s \mid \varphi(N)$, y por lo tanto $[a^{\varphi(N)}] = [1]$ como queriamos probar. □

Demostración clásica. Sea $C = \{1 \leq x \leq N-1 \,:\, \gcd(x, N) = 1\}$. Como a es coprimo con N, y en particular es invertible módulo N, resulta que la aplicación $m_a : C \to C$ dada por $m_a(x) = ax \bmod N$ es una biyección. Entonces

$$\prod_{x \in S} ax \equiv \prod_{x \in S} x \pmod{N}$$

y por lo tanto

$$a^{\varphi(N)} \prod_{x \in S} x \equiv \prod_{x \in S} x \pmod{N}.$$

Como todos los elementos de S son invertibles módulo N, podemos quitarlos de ambos miembros de la identidad de arriba, y entonces $a^{\varphi(N)} \equiv 1 \pmod{N}$. □

Problema 4.4. Mostrar que $\varphi(100) = 40$. De acuerdo con el teorema de Euler-Fermat se tiene que $a^{40} \equiv 1 \pmod{100}$ para cualquier a coprimo con 100. Escribir un programa en Python3 que verifique esa afirmación. Calcular la estructura del grupo $(\mathbb{Z}/100\mathbb{Z})^*$ determinando manualmente la estructura de $(\mathbb{Z}/4\mathbb{Z})^*$ y $(\mathbb{Z}/25\mathbb{Z})^*$. Demostrar que $a^{20} \equiv 1 \pmod{100}$ para todo a coprimo con 100 y modificar el programa anterior para verficar esta nueva afirmación.

Problema 4.5. Calcular (a mano) el resto de dividir $10^{10000000}$ por $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$ y luego verificarlo haciendo `(10 ** 10000000) % (2*3*5*7*11*13)` en Python3. Comparar la velocidad de esta instrucción con `potencia_mod(10, 10000000, 2*3*5*7*11*13)`.

Problema 4.6. Un entero $N \geq 2$ se dice pseudoprimo de Fermat en base a, para $a \in \mathbb{Z}$ coprimo con N, si $a^{N-1} \equiv 1 \pmod{N}$. Se dice que N es pseudoprimo de Fermat fuerte si es pseudoprimo en cualquier base a coprimo con N. El teorema de Euler-Fermat muestra que todo primo es pseudoprimo de Fermat fuerte, pero lamentablemente la recíproca no es cierta. Los números compuestos $N \geq 2$ que son pseudoprimos de Fermat fuertes se llaman números de Carmichael. Escribir un programa en Python3 que determine los 10 primeros números de Carmichael.

En lo que sigue vamos a estudiar en detalle la estructura de los grupos de unidades $(\mathbb{Z}/N\mathbb{Z})^*$.

Recordemos primero que todo grupo abeliano finitamente generado G es isomorfo a

$$\mathbb{Z}^r \oplus \mathbb{Z}/d_1\mathbb{Z} \oplus \mathbb{Z}/d_2\mathbb{Z} \oplus \cdots \oplus \mathbb{Z}/d_k\mathbb{Z}$$

para un cierto $r \geq 0$ (el rango del grupo) y ciertos $d_1, \ldots, d_k \geq 1$ tales que $d_i \mid d_{i+1}$ para todo i. Lógicamente, en la escritura de arriba, cada uno de los sumandos $\mathbb{Z}/d_i\mathbb{Z}$ es visto como grupo abeliano *aditivo* y no como anillo. Para el caso de grupos abelianos finitos, se tiene que $r = 0$. Si además el grupo es cíclico, se tiene $k = 1$, es decir, $G \simeq \mathbb{Z}/d_1\mathbb{Z}$ para cierto $d_1 \geq 1$.

El orden de un elemento a en un grupo G es el menor $n \geq 1$ tal que

$$n \cdot a = \underbrace{a + a + \cdots + a}_{n \text{ veces}} = 0_G,$$

o equivalentemente, la cantidad de elementos del subgrupo generado por a. Por el teorema de Lagrange, se tiene siempre que $ord(a) \mid |G|$. En el caso del grupo multiplicativo $(\mathbb{Z}/N\mathbb{Z})^*$, el orden de un elemento $[a]_N$ es el menor $n \geq 1$ tal que $a^n \equiv 1 \pmod{N}$, y como mencionamos arriba, se tiene que

$ord([a]_N) \mid \varphi(N)$. Por abuso de notación, en los casos donde el valor de N sea claro por el contexto, escribiremos $ord(a)$ en lugar de $ord([a]_N)$. Por supuesto, esto solo tiene sentido cuando $\gcd(a, N) = 1$.

Ahora estamos en condiciones de entender la estructura de los grupos $(\mathbb{Z}/N\mathbb{Z})^*$ completamente. Por supuesto, basta con reducirse al caso $N = p^n$ para $p \geq 2$ primo y $n \geq 1$.

Teorema 4.3. *Sea $p \geq 2$ primo y $n \geq 1$. Entonces $(\mathbb{Z}/p^n\mathbb{Z})^*$ es cíclico para cualquier p impar y para el caso $p = 2$, $n \leq 2$, es decir, es isomorfo a $\mathbb{Z}/\varphi(p^n)\mathbb{Z}$. Para $n \geq 3$, el grupo $(\mathbb{Z}/2^n\mathbb{Z})^*$ es isomorfo a $\mathbb{Z}/2\mathbb{Z} \oplus \mathbb{Z}/2^{n-2}\mathbb{Z}$.*

La demostración del teorema 4.3 es larga y la vamos a dividir en varias partes.

Proposición 4.4. *Sea $p \geq 2$ primo. Entonces $(\mathbb{Z}/p\mathbb{Z})^*$ es cíclico.*

Proof. Supongamos que $(\mathbb{Z}/p\mathbb{Z})^*$ es isomorfo a $\mathbb{Z}/d_1\mathbb{Z} \oplus \cdots \mathbb{Z}/d_k\mathbb{Z}$ para ciertos $d_1, \ldots, d_k \geq 2$ tales que $d_i \mid d_{i+1}$ para todo $i = 1, \ldots, k-1$. Esto implica que $x^{d_k} = 1$ para todo $x \in (\mathbb{Z}/p\mathbb{Z})^*$, o equivalentemente, que el polinomio $X^{d_k} - 1 \in (\mathbb{Z}/p\mathbb{Z})[X]$ tiene como raíces a todos los elementos de $(\mathbb{Z}/p\mathbb{Z})^*$. Como sabemos que $\mathbb{Z}/p\mathbb{Z}$ es un cuerpo, el polinomio puede tener como mucho d_k raíces, y entonces $d_k \geq \varphi(p) = p - 1$. Por otra parte, sabemos que $d_1 d_2 \cdots d_k = \varphi(p) = p - 1$, y entonces $d_k \leq p - 1$. La única forma de que esto es posible es si $k = 1$ y $d_1 = p - 1$, es decir, el grupo $(\mathbb{Z}/p\mathbb{Z})^*$ es isomorfo al grupo cíclico $\mathbb{Z}/(p-1)\mathbb{Z}$. □

Lema 4.5. *Sean $p \geq 3$ primo, $n \geq 1$ y $b = 1 + p^n u$ para cierto $u \in \mathbb{Z}$ tal que $p \nmid u$. Entonces $b^p = 1 + p^{n+1} v$ con $p \nmid v$.*

Proof. Usando la fórmula del binomio, nos queda

$$b^p = (1 + p^n u)^p = \sum_{i=0}^{p} \binom{p}{i} p^{ni} u^i = 1 + p^{n+1} u + \sum_{i=2}^{p} \binom{p}{i} p^{ni} u^i = 1 + p^{n+1} u + p^{n+2} t$$

ya que $p^{n+2} \mid \binom{p}{i} p^{ni}$ para cualquier $i \geq 2$. La conclusión se sigue de tomar $v = u + pt$. □

Lema 4.6. *Sean $p \geq 3$ primo y $a \in \mathbb{Z}$ tal que $p \nmid a$ y $a^{p-1} \not\equiv 1 \pmod{p^2}$. Si $ord([a]_p) = p - 1$, entonces $ord([a]_{p^n}) = (p-1)p^{n-1}$ para todo $n \geq 2$.*

Proof. Por el teorema de Euler-Fermat sabemos que $a^{p-1} \equiv 1 \pmod{p}$, es decir, $a^{p-1} = 1 + pu$ para cierto $u \in \mathbb{Z}$. Por otra parte, debe ser $p \nmid u$, ya que de lo contrario tendríamos que $a^{p-1} \equiv 1 \pmod{p^2}$, en contradicción con las hipótesis.
Si $a^\ell \equiv 1 \pmod{p^n}$, también será que $a^\ell \equiv 1 \pmod{p}$ y por lo tanto $a^{\gcd(\ell, p-1)} \equiv 1 \pmod{p}$. Teniendo en cuenta que $ord([a]_p) = p - 1$, la única posibilidad es que $p - 1 \mid \ell$. En particular, esto prueba que $ord([a]_{p^n})$ es un múltiplo de $p-1$. Como también sabemos que $ord([a]_{p^n})$ divide a $\varphi(p^n) = (p-1)p^{n-1}$, nos queda que $ord([a]_{p^n}) = (p-1)p^k$ para cierto $1 \leq k \leq n-1$.
Aplicando k veces consecutivas el lema 4.5 al entero $b = a^{p-1} = 1 + pu$, nos queda que $a^{(p-1)p^k} = 1 + p^{k+1} v$ para cierto $v \in \mathbb{Z}$ tal que $p \nmid v$. Por otra parte, debe ser $a^{(p-1)p^k} \equiv 1 \pmod{p^n}$ y por lo tanto $k = n - 1$ como queríamos probar. □

Proposición 4.7. *Sean $p \geq 3$ primo y $n \geq 1$. Entonces $(\mathbb{Z}/p^n\mathbb{Z})^*$ es cíclico.*

Proof. En la proposición 4.4 ya vimos el caso $n = 1$, es decir, ya sabemos que $(\mathbb{Z}/p\mathbb{Z})^*$ es cíclico. Tomemos $a \in \mathbb{Z}$ tal que $p \nmid a$ y que $[a]_p$ sea un generador de $(\mathbb{Z}/p\mathbb{Z})^*$, es decir, que $ord([a]_p) = p - 1$. Por supuesto, tenemos que $a^{p-1} = 1 + pu$ para cierto $u \in \mathbb{Z}$. En el caso de que $p \nmid u$, el lema 4.6, prueba que $[a]_{p^n}$ es un generador de $(\mathbb{Z}/p^n\mathbb{Z})^*$ para todo $n \geq 2$.
En el caso de que $p \mid u$, es decir, que $a^{p-1} \equiv 1 \pmod{p^2}$, podemos cambiar a a por $a + p$ y volver a empezar. En efecto, se puede ver que

$$(a + p)^{p-1} = 1 + pu + (p-1)pa^{p-2} + \sum_{i=2}^{p-1} \binom{p-1}{i} a^{p-1-i} p^i$$

es de la forma $1 + pu'$ con $p \nmid u'$, ya que pu y todos lo términos con $i \geq 2$ son múltiplos de p^2, pero el término $(p-1)a^{p-2}p$ es divisible por p pero no p^2. □

Un entero $a \in \mathbb{Z}$ tal que $p \nmid a$ y $ord([a]_{p^n}) = \varphi(p^n)$ se llama una raíz primitiva módulo p^n. La proposición anterior muestra la existencia de raíces primitivas para p primo impar y $n \geq 1$.

Lema 4.8. *Sea $n \geq 2$. Entonces $ord([5]_{2^n}) = 2^{n-2}$.*

Proof. Sabemos que $ord([5]_{2^n}) \mid \varphi(2^n) = 2^{n-1}$ y por lo tanto será de la forma 2^k para cierto $k \geq 0$. Vamos a probar inductivamente que $5^{2^k} = 1+2^{k+2}u_k$ con u_k impar. Para $k = 0$ es inmediato tomando $u_0 = 1$. Ahora veamos el caso $k+1$ suponiendo probado el caso k.

$$5^{2^{k+1}} = (1 + 2^{k+2}u_k)^2 = 1 + 2^{k+3}u_k + 2^{2k+4}u_k^2 = 1 + 2^{k+3}(u_k + 2^{k+1}u_k^2)$$

Basta con definir $u_{k+1} = u_k + 2^{k+1}u_k^2$. Esto completa la inducción. De este resultado, se puede ver que el menor $k \geq 0$ que hace que $5^{2^k} \equiv 1 \pmod{2^n}$ es $k = n-2$, como queríamos probar. □

Lema 4.9. *Sea $n \geq 3$ y sea $a \in \mathbb{Z}$ impar. Entonces $ord([a]_{2^n}) \mid 2^{n-2}$.*

Proof. El enunciado es equivalente a decir que $a^{2^{n-2}} \equiv 1 \pmod{2^n}$ para todo $n \geq 3$ y a impar. Procederemos por inducción. El caso $n = 3$ es inmediato, ya que $(1 + 2r)^2 = 1 + 4r + 4r^2 = 1 + 4r(r + 1) \equiv 1 \pmod 8$ para cualquier $r \in \mathbb{Z}$. Para al paso inductivo, supongamos que ya tenemos el caso n probado, es decir, que $a^{2^{n-2}} = 1 + 2^n u$ para cierto $u \in \mathbb{Z}$. Elevando al cuadrado nos queda $a^{2^{n-1}} = (1 + 2^n u)^2 = 1 + 2^{n+1}u + 2^{2n}u^2 \equiv 1 \pmod{2^{n+1}}$, lo que demuestra el caso $n + 1$. □

Demostración del teorema 4.3. El caso p impar fue probado en la proposición 4.7, por lo que solo falta ver el caso $p = 2$. Los casos $n = 1$ y $n = 2$ se puede comprobar manualmente. Para $n \geq 3$, digamos que $(\mathbb{Z}/2^n\mathbb{Z})^* \simeq \mathbb{Z}/d_1\mathbb{Z} \oplus \cdots \oplus \mathbb{Z}/d_k\mathbb{Z}$ con $d_i \mid d_{i+1}$ para $i = 1, 2, \ldots, k-1$. El lema 4.9 demuestra que $d_k \mid 2^{n-2}$ y el lema 4.8 que $2^{n-2} \mid d_k$. En particular, debe ser $d_k = 2^{n-2}$. Por otra parte, tenemos que $d_1 \cdots d_k = \varphi(2^n) = 2^{n-1}$. La única forma de que esto se cumpla es si $k = 2$ y $d_1 = 2$ y por lo tanto $(\mathbb{Z}/2^n\mathbb{Z})^* \simeq \mathbb{Z}/2\mathbb{Z} \oplus \mathbb{Z}/2^{n-2}\mathbb{Z}$. □

En el caso p primo, sabemos que $(\mathbb{Z}/p\mathbb{Z})^* \simeq \mathbb{Z}/(p-1)\mathbb{Z}$. El isomorfismo (de grupos) se puede escribir de forma explícita: tomamos $a \in \mathbb{Z}$ una raíz primitiva módulo p y definimos

$$\exp : \mathbb{Z}/(p-1)\mathbb{Z} \to (\mathbb{Z}/p\mathbb{Z})^*$$
$$[k]_{p-1} \mapsto [a^k]_p$$

El morfismo inverso $\log : (\mathbb{Z}/p\mathbb{Z})^* \to \mathbb{Z}/(p-1)\mathbb{Z}$ se llama *logaritmo discreto* (en base a y módulo p si es necesario aclarar). El programa 2 muestra que $\exp([k])$ puede calcularse eficientemente. Sin embargo, no se conoce actualmente ningún algoritmo de complejidad binaria polinomial para calcular logaritmos discretos. Esta particularidad se usa extensivamente en protocolos criptográficos.

A continuación vamos a ver tres aplicaciones interesantes de las raíces primitivas: el test de primalidad de Lucas, el criterio de Korselt para números de Carmichael y el método de factorización $p-1$ de Pollard.

Teorema 4.10 (Test de primalidad de Lucas)**.** *Para cualquier $N \geq 2$ son equivalentes:*

1. *N es primo.*
2. *Para todo primo p que divide a $N-1$, existe $a \in \mathbb{Z}$ tal que $a^{N-1} \equiv 1 \pmod N$ y $a^{(N-1)/p} \not\equiv 1 \pmod N$.*

Proof. $(1 \Rightarrow 2)$: Elijamos $[a]$ un generador de $(\mathbb{Z}/N\mathbb{Z})^*$, es decir, a es una raíz primitiva módulo N. Por el teorema de Euler-Fermat sabemos que $a^{N-1} \equiv 1 \pmod N$ y por ser raíz primitiva $N-1$ es el menor exponente i tal que $a^i \equiv 1 \pmod N$. En particular, $a^{(N-1)/p} \not\equiv 1 \pmod N$ para cualquier $p \mid N-1$.
$(2 \Rightarrow 1)$: Sea p un primo que divide a $N-1$ y sea a tal que $a^{N-1} \equiv 1 \pmod N$ y $a^{(N-1)/p} \not\equiv 1 \pmod N$. Esto nos dice que el orden multiplicativo de $[a]$ en $(\mathbb{Z}/N\mathbb{Z})^*$ es un divisor de $N-1$ pero no de $(N-1)/p$. Esto implica que la máxima potencia de p que divide a $N-1$ tiene que dividir al orden de $[a]$ y en particular a $\varphi(N) = |(\mathbb{Z}/N\mathbb{Z})^*|$. Repitiendo esto para todos los primos p que dividen a $N-1$, concluímos que $N-1 \mid \varphi(N)$ y por lo tanto $\varphi(N) = N-1$. Esto solo es posible si N es primo. □

Teorema 4.11 (Criterio de Korselt)**.** *Un entero $N \geq 2$ es pseudoprimo fuerte de Fermat (ver problema 4.6) si y solo si N es libre de cuadrados (todos los factores primos de N tienen multiplicidad 1) y $p-1 \mid N-1$ para todo factor primo p de N.*

Proof. Supongamos que $N = p_1 p_2 \cdots p_k$ con los p_i primos distintos y tal que $p_i - 1 \mid N-1$ para $i = 1, \ldots, k$. Sea $a \in \mathbb{Z}$ coprimo con N, es decir, $p_i \nmid a$ para todo i. Por el teorema de Euler-Fermat tenemos que $a^{p_i - 1} \equiv 1 \pmod{p_i}$ y como $p_i - 1 \mid N-1$ también $a^{N-1} \equiv 1 \pmod{p_i}$ para todo i. Por lo tanto $a^{N-1} \equiv 1 \pmod{N}$, es decir, N es pseudoprimo de Fermat en base a. Como esto vale para cualquier a, concluímos que N es un pseudoprimo de Fermat fuerte.
Ahora veamos la recíproca. Supongamos que $N = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ es pseudoprimo de Fermat fuerte, donde los p_i son primos distintos y los $n_i \geq 1$. Como N debe ser pseudoprimo en base $a = -1$, tenemos que $(-1)^{N-1} \equiv 1 \pmod{N}$ y por lo tanto N debe ser impar o bién $N = 2$, en cuyo caso ya hemos terminado. A partir de ahora supondremos que N es impar. En primer lugar, tenemos que probar que los n_i no pueden ser mayores que 1. Supongamos que $n_i \geq 2$ para algún i y tomemos a una raíz primitiva módulo p_i^2. Por hipótesis sabemos que $a^{N-1} \equiv 1 \pmod{p_i^2}$ y por ser a primitiva, debe ser $\varphi(p_i^2) \mid N-1$. Pero esto es imposible, ya que $\varphi(p_i^2) = p_i(p_i - 1)$ es divisible por p_i pero $N-1$ no lo es. Esta contradicción muestra que $n_1 = n_2 = \cdots = n_k = 1$, es decir, que N es libre de cuadrados. Por último, tomemos a una raíz primitiva módulo p_i. Como $a^{N-1} \equiv 1 \pmod{p_i}$ por hipótesis y a es primitiva, tenemos también que $p_i - 1 \mid N-1$ como queríamos probar. □

Definición 4.12. Sea $B \geq 2$. Un entero $n \neq 0$ se dice B-suave (B-smooth) si solo es divisible por primos menores o iguales que B.

Supongamos que tenemos un entero $N \geq 2$ compuesto al que queremos factorizar como producto de dos factores no triviales. El método $p-1$ de Pollard permite hacer esto suponiendo que N tiene al menos dos factores primos p y q distintos, tales que $p-1$ es B-suave (para un $B \geq 2$ conocido) y que $q-1$ no es B-suave. Si llamamos

$$\beta = \prod_{\substack{p \text{ primo} \\ p \leq B}} p^{\lceil \log_p N \rceil}$$

la condición de que $p-1$ es B-suave es equivalente a decir que $p-1 \mid \beta$. El método de Pollard es el siguiente: se elige un $a \in [1, N-1]$ uniformemente al azar, se comprueba que $x = \gcd(a, N) = 1$ y finalmente se calcula $y = \gcd(a^\beta - 1, N)$. En caso de que $x \neq 1$, ya hemos resuelto el problema, ya que $x \mid N$ y $1 < x \leq a \leq N-1$. Queda por ver que sucede en el caso $x = 1$, es decir, cuando a es coprimo con N. Como a será también coprimo con p, tenemos que $a^{p-1} \equiv 1 \pmod{p}$ por el teorema de Euler-Fermat, y por lo tanto $a^\beta \equiv 1 \pmod{p}$. Esto nos dice que $y = \gcd(a^\beta - 1, N)$ será divisible por p. Veamos que es bastante improbable que y sea divisible por q. En efecto, la cantidad de soluciones de la congruencia $a^\beta \equiv 1 \pmod{q}$ con $\gcd(a, q) = 1$ es $\gcd(\beta, q-1) < \frac{q-1}{B}$, ya que hay al menos un factor primo de $q-1$ mayor que B y que por lo tanto no aparece en β. Eso nos dice que la probabilidad condicional de que $a^\beta \equiv 1 \pmod{q}$ sabiendo que $\gcd(a, N) = 1$ es menor que $1/B$. En caso de que y no produzca un factor no trivial, es decir, que $y = N$, bastará con repetir el experimento.

Problema 4.7. Implementar la función `menor_raiz_primitiva(p)` que obtenga la menor raíz primitiva $1 \leq a \leq p-1$ para un primo p dado.

Problema 4.8. Calcular la estructura de $(\mathbb{Z}/N\mathbb{Z})^*$ para $N = 100, 1000, 10!$. ¿Cuál es el menor n tal que $a^n \equiv 1 \pmod{N}$ para todo a coprimo con N?

Problema 4.9. Encontrar todos los elementos de $(\mathbb{Z}/2^n\mathbb{Z})^*$ de orden 2. Demostrar que la aplicación

$$\begin{aligned} \mathbb{Z}/2\mathbb{Z} \oplus \mathbb{Z}/2^{n-2}\mathbb{Z} &\to (\mathbb{Z}/2^n\mathbb{Z})^* \\ ([a], [b]) &\mapsto [(-1)^a 5^b] \end{aligned}$$

es un isomorfismo de grupos para todo $n \geq 3$.

Problema 4.10. Demostrar que en un grupo cíclico finito G hay $\varphi(|G|)$ elementos de orden $|G|$. ¿Cuántas raíces primitivas hay módulo un primo p?

Problema 4.11. Para cada uno de los primeros 10 primos $p = 2, 3, 5, 7, \ldots$ determinar el orden multiplicativo de 2 módulo p y obtener condiciones necesarias y suficientes sobre n para que $p \mid 3 \cdot 2^n + 1$.

Problema 4.12. Mostrar, con la ayuda de Python3, que $N = 3 \cdot 2^{189} + 1$ es primo utilizando el test de primalidad de Lucas. Para cada $p \in \{2, 3\}$, determinar la cantidad de enteros $1 \leq a < N$ tales que $a^{(N-1)/p} \not\equiv 1 \pmod{N}$. ¿Es posible encontrar a tal que $a^{(N-1)/2} \not\equiv 1 \pmod{N}$ y $a^{(N-1)/3} \not\equiv 1 \pmod{N}$?

Problema 4.13. Sea p primo y sea $e \geq 1$ tal que $\gcd(e, p-1) = 1$. Demostrar que la aplicación

$$\begin{aligned} (\mathbb{Z}/p\mathbb{Z})^* &\to (\mathbb{Z}/p\mathbb{Z})^* \\ [x] &\mapsto [x^e] \end{aligned}$$

es un isomorfismo de grupos.

Problema 4.14. ¿Cuántas soluciones $(x, y) \in \mathbb{Z}/p\mathbb{Z}$ tiene la ecuación $y^2 \equiv x^3 + 7 \pmod{p}$ para un primo $p \equiv 2 \pmod{3}$? Modificar el programa `curva_eliptica_mod13.py` del tema #1 para comprobarlo para los primeros 100 primos.

Problema 4.15. Utilizar el criterio de Korselt para probar que si $6k+1$, $12k+1$ y $18k+1$ son primos (para un cierto entero $k \geq 1$), entonces $N = (6k+1)(12k+1)(18k+1)$ es un número de Carmichael. Escribir un programa en Python3 para encontrar los primeros 5 números de Carmichael de esta forma.

Problema 4.16. Utilizar el método $p-1$ de Pollard para factorizar

$$N = 153363929807872895964194606676600755969807832413692875733080902984835973327176 7$$

utilizando $B = 50$. ¿Cómo se puede calcular $\gcd(a^\beta - 1, N)$ de forma eficiente?