Inverso modular

Profesor: Andrés Abeliuk
Auxiliares: Blaz Korecic, Diego Arias, Dmitri Ramirez, Javier Oliva y Vicente Villarroel

Introducción

En aritmética modular, el inverso modular a^{-1} de un número a en módulo p es un número tal que al multiplicarlo con a se obtiene el neutro multiplicativo. Es decir:

$$a^{-1}a \equiv 1 \pmod{p}$$

En otras palabras, multiplicar por el inverso modular es el análogo a «dividir» un número en módulo p.

Cabe notar que el inverso modular de un número solo existe si a y p son coprimos. En este curso, principalmente nos enfocaremos en los casos en los que p es primo, por lo que basta con determinar que a no es un múltiplo de p para saber que son coprimos.

¿Pero cómo podemos calcular el inverso modular?

Calculando el inverso modular

El Pequeño Teorema de Fermat

Sea p primo, $a \in \mathbb{Z}$, a y p coprimos. Se tiene que:

$$a^p \equiv a \pmod{p}$$

Por brevedad, se omitirá la demostración en este documento, pero hay muchas demostraciones de este teorema en otras partes.

A partir de esto, se tiene que:

$$a^{p-2} = a^{-1} \pmod{p}$$

Luego, si queremos calcular a^{-1}, basta con calcular a^{p-2}.

Calculando el exponente

Un primer comienzo para intentar calcular a^{p-2} es multiplicar a consigo mismo $p-2$ veces. Sin embargo, este método es bastante ineficiente, pues requiere hacer $O(p)$ multiplicaciones. Considerando que p puede ser un número muy grande, nos interesa encontrar una forma más fácil de calcularlo.

Si $p-2$ fuera una potencia de 2, podríamos calcular a^2 primero, luego calcular $\left(a^2\right)^2$, después calcular $\left(\left(a^2\right)^2\right)^2$ y así sucesivamente hasta llegar a $p-2$, por lo que podríamos calcularlo en $O(\log p)$.

Sin embargo, este no siempre es el caso (de hecho, como p es primo, $p-2$ es impar o 0, por lo que nunca es el caso). Pero sí nos da un buen primer avance para el problema.

Exponenciación binaria

Sabemos que podemos calcular a^{2^n} rápidamente, y sabemos que $a^{b+c} = a^b a^c$. Luego, podríamos representar $p-2$ como una suma de potencias de 2. Así, podríamos calcular cada una de ellas rápidamente, y a partir de ello calcular a^{p-2}.

Afortunadamente, hay una herramienta que conocemos que nos permite separar $p-2$ en potencias de 2. Si escribimos $p-2$ en su representación binaria, efectivamente estamos escribiendo el número como una suma de potencias de 2. Por ejemplo:

$11_{10} = 8_{10} + 2_{10} + 1_{10} = 1011_2$

Por lo tanto, para calcular a^{11}, basta con calcular a^1, a^2 y a^8 utilizando el método antes descrito y multiplicarlos entre sí.

Entonces, lo que podríamos hacer es calcular a elevado a cada potencia de 2 utilizando el calculo anterior, y si $p-2$ contiene un 1 en su representación binaria en la posición respectiva, multiplicamos la respuesta por dicha potencia. Es decir:

- Inicializamos la respuesta como 1 (debido a que $a^0 = 1$), y consideramos actual $= a^1$.
- Para cada dígito de b de derecha a izquierda:
  - Si b tiene un 1 en dicha posición, multiplicamos la respuesta por el valor actual.
  - Tomamos el valor actual como el siguiente valor de a^{2^n}.
- Retornamos la respuesta

Tomando el ejemplo anterior:

- Primero consideramos actual $= c = a$, respuesta = $r = 1$.
- Luego, como $11_{10} = 1011_2$, su dígito de más a la derecha es un 1, por lo que multiplicamos r por c. Luego, $r = a$.
- Ahora, actualizamos c como c^2, es decir, $c = a^2$.
- Le quitamos el dígito de la derecha a b y nos queda $b = 101_2$.
- Como su primer dígito es un 1, $r = rc = aa^2 = a^3$
- Actualizamos $c = c^2 = a^4$
- Le quitamos el dígito de la derecha a b y nos queda $b = 10_2$
- Como su dígito de la derecha no es un 1, no actualizamos la respuesta
- Actualizamos $c = c^2 = a^8$
- Le quitamos el dígito de la derecha a b y nos queda $b = 1_2$.
- Como su primer dígito es un 1, $r = rc = a^3 a^8 = a^{11}$
- Actualizamos $c = c^2 = a^4$
- Como ya recorrimos todos los dígitos de b, retornamos r.

A continuación, se encuentra una implementación de este algoritmo en C++.

```
// calcula a^b mod p
int modpow(int a, int b, int p) {
    // valor inicial, a^0=1
    int ans = 1;
    int actual = a;
    // vamos revisando b digito por digito
    while (b > 0) {
        // si el dígito actual es 1, debemos multiplicar la respuesta por a^2^n
        if (b % 2 == 1)
            ans = (ans * actual) % p;
        // calculamos c = a^2^(n+1) como a^2^n * a^2^n
        actual = (actual * actual) % p;
        // le quitamos el ultimo digito a b
        b /= 2;
    }
    return ans;
}

// calcula el inverso modular de a mod p
int invmod(int a, int p) {
    // por el PTF, a^(p-2)=a^-1
    return modpow(a, p-2,p);
}
```

Cabe destacar que es importante aplicar módulo p a cada una de las multiplicaciones que realizamos, ya que de lo contrario, como los exponentes crecen muy rápido, esto rápidamente haría overflow. Luego, vale la pena preguntarse ¿Cuándo el código actual podría hacer overflow? ¿Qué tan grandes pueden ser a y p de forma de que no tengamos problemas?