Aritmética modular

Sebastian Mestre

¿Qué es la aritmética modular?

- Sistema de numeración construido sobre el **resto** de la división.
- Dos números son **congruentes módulo** m si su diferencia es un múltiplo de m.

Definición

Dados enteros a, b y $m \geq 1$,

$$a \equiv b \pmod{m} \quad \Leftrightarrow \quad a - b = k \cdot m \text{ para algún } k \in \mathbb{Z}.$$

- En particular, sumar m no cambia la clase:

$$a + m \equiv a \pmod{m}.$$

Propiedades de la congruencia

- La relación $\equiv \pmod{m}$ es una **relación de equivalencia**:
  - Reflexiva: $a \equiv a \pmod{m}$.
  - Simétrica: si $a \equiv b \pmod{m}$, entonces $b \equiv a \pmod{m}$.
  - Transitiva: si $a \equiv b \pmod{m}$ y $b \equiv c \pmod{m}$, entonces $a \equiv c \pmod{m}$.
- Se comporta bien con las operaciones básicas:

$$\begin{array}{l} a \equiv b \pmod{m} \\ c \equiv d \pmod{m} \end{array} \quad \Rightarrow \quad \begin{cases} a + c \equiv b + d \pmod{m}, \\ a - c \equiv b - d \pmod{m}, \\ a \cdot c \equiv b \cdot d \pmod{m}. \end{cases}$$

Clases de equivalencia

- Para cada entero a, consideramos el conjunto de todos los números congruentes con a módulo m.
- Este conjunto se llama **clase de equivalencia** de a módulo m:

$$[a] = \{\, b \in \mathbb{Z} \mid b \equiv a \pmod{m} \,\}.$$

Ejemplo: $m = 5$

$$[0] = \{\ldots, -5, 0, 5, 10, 15, \ldots\}$$
$$[1] = \{\ldots, -4, 1, 6, 11, 16, \ldots\}$$
$$[2] = \{\ldots, -3, 2, 7, 12, 17, \ldots\}$$
$$[3] = \{\ldots, -2, 3, 8, 13, 18, \ldots\}$$
$$[4] = \{\ldots, -1, 4, 9, 14, 19, \ldots\}$$

Notar que $[5] = [0]$, es la misma clase.

El conjunto $\mathbb{Z}_m$

- Hay exactamente m clases de equivalencia módulo m:

$$[0], [1], \ldots, [m-1].$$

- Al conjunto de estas clases se lo denota típicamente por $\mathbb{Z}_m$:

$$\mathbb{Z}_m = \{[0], [1], \ldots, [m-1]\}.$$

- Como la suma, la resta y el producto **respetan** la congruencia, podemos definir operaciones entre clases:

$$[x] + [y] = [x + y], \quad [x] - [y] = [x - y], \quad [x] \cdot [y] = [x \cdot y].$$

Representación canónica

- Para todo entero x, existe un único entero $r \in [0, m-1]$ tal que

$$x \equiv r \pmod{m}.$$

- A este r lo llamamos **representación canónica** de x módulo m.
- Si x es positivo, r es simplemente el resto de dividir x por m.
- Es muy cómodo trabajar siempre con los números en su representación canónica.

Cuidado con % en C++

- En C++, cuando x es negativo, `x % m` **no** da la representación canónica (puede ser negativa).
- Podemos corregirlo de forma estándar:

Función de representación canónica

```
1 int const m = 1000000007; // muy importante que sea constante
2
3 int canonica(int x) {
4   return (x % m + m) % m;
5 }
6
```

Operaciones en representación canónica

- Si mantenemos todos los números en $[0, m-1]$, las operaciones básicas se pueden implementar de forma segura.
- Truco: una versión más rápida que `canonica` cuando sabemos que el resultado está en $[0, 2m-1]$.

Implementación

```
1 // igual que canonica en el intervalo [0, 2m-1], y m s r pido
2 int nm(int x) {  return x - m * (x >= m); }
3
4 int add(int x, int y) { return nm(x + y); }
5
6 int sub(int x, int y) {  return nm(x + m - y);}
7
8 // cuidado: x * y puede desbordar int, usamos long long
9 int mul(int x, int y) {  return (long long)x * y % m; }
```

Potenciación modular

- Necesitamos calcular x^y mód m muchas veces en programación competitiva.
- Usamos **exponenciación binaria** para obtener costo $O(\log y)$.

Implementación

```
1 // O(log y)
2 int pot(int x, int y) {
3   if (y == 0) return 1;
4   if (y % 2 == 0) return pot(mul(x, x), y / 2);
5   return mul(x, pot(x, y - 1));
6 }
7
```

Inverso multiplicativo

- Si m es primo, podemos usar el **pequeño teorema de Fermat**:

$$x^{m-1} \equiv 1 \pmod{m} \quad \Rightarrow \quad x^{m-2} \equiv x^{-1} \pmod{m}.$$

- Entonces, el inverso multiplicativo se puede calcular en $O(\log m)$.

Implementación

```
1 int inv(int x) {
2   return pot(x, m - 2);
3 }
4
```

Trucos con inversos y factoriales

- A veces queremos todos los inversos $1, 2, \ldots, n$ en $O(n)$.
- También es común precomputar factoriales y sus inversos para combinatoria.

Implementaciones típicas

```
int inv[n + 1];
inv[1] = 1;
for (int i = 2; i <= n; i++) {
  inv[i] = mul(inv[m % i], m - m / i);
}

int fac[n + 1], ifac[n + 1];
fac[0] = 1;
forn(i, n) fac[i + 1] = mul(fac[i], i + 1);  // factoriales
ifac[n] = inv(fac[n]);
dforn(i, n) ifac[i] = mul(ifac[i + 1], i + 1); // inversos factoriales
forr(i, 1, n) inv[i] = mul(ifac[i], fac[i - 1]); // inversos 1..n

```

Ideas clave

- La aritmética modular trabaja con clases de equivalencia definidas por el resto módulo m.
- Mantener siempre la **representación canónica** simplifica las implementaciones y evita errores.
- Con operaciones básicas (`add`, `sub`, `mul`, `pot`, `inv`) podemos resolver muchos problemas de combinatoria y conteo.

Preguntas

Q&A