**Bitmasking** (enmascaramiento de bits) es una técnica que consiste en **usar los bits de un número entero para representar un conjunto o un estado**.

* Un número binario de n bits puede representar todas las combinaciones de un conjunto de tamaño n.
* Por ejemplo, el número 13 en binario es 1101, lo que puede significar:
  + El elemento 0 está presente ✅ (bit 0 = 1)
  + El elemento 1 no está ❌ (bit 1 = 0)
  + El elemento 2 está ✅ (bit 2 = 1)
  + El elemento 3 está ✅ (bit 3 = 1)

Así, 13 representa el conjunto {0, 2, 3}.

**💡 Ideas fundamentales**

| **Operación** | **Explicación** | **Ejemplo (A=1010₂, B=0110₂)** | **Resultado** |
| --- | --- | --- | --- |
| A & B | AND bit a bit | 1010 & 0110 | 0010 (2) |
| `A | B` | OR bit a bit | `1010 |
| A ^ B | XOR bit a bit | 1010 ^ 0110 | 1100 (12) |
| ~A | NOT bit a bit | ~1010 | 0101 (inversión) |
| A << k | Desplaza bits a la izquierda (×2^k) | 1010 << 1 | 10100 (20) |
| A >> k | Desplaza bits a la derecha (÷2^k) | 1010 >> 1 | 0101 (5) |

👉 En C++:

(1 << i) // número con solo el bit i encendido

mask | (1 << i) // enciende el bit i

mask & ~(1 << i) // apaga el bit i

mask ^ (1 << i) // invierte el bit i

mask & (1 << i) // comprueba si el bit i está encendido

**⚙️ Lógica**

**1️⃣ Representación de conjuntos**

Podemos usar un int o long long para representar subconjuntos.

int mask = 0;

mask |= (1 << 2); // añade elemento 2

mask |= (1 << 4); // añade elemento 4

cout << mask; // imprime 20 (10100₂)

**2️⃣ Iterar sobre todos los subconjuntos**

int n = 4;

for (int mask = 0; mask < (1 << n); mask++) {

cout << "Subconjunto: ";

for (int i = 0; i < n; i++) {

if (mask & (1 << i))

cout << i << " ";

}

cout << endl;

}

👉 Este código recorre los 2^n subconjuntos posibles.

**3️⃣ Submáscaras**

Dado un mask, puedes recorrer todos sus submáscaras:

for (int sub = mask; sub; sub = (sub - 1) & mask)

cout << sub << " ";