



Manipulación de Bits

Manipulación de Bits

- ❑ La mayoría de mejoras en un programa son de “alto nivel” (afectan al algoritmo en lugar de la implementación).
- ❑ La manipulación de bits es una de las mejoras a “bajo nivel” más conocidas, permite que nuestro programa sea algo más rápido, simplifica nuestro código e incluso podría mejorar la complejidad de nuestro algoritmo.



Operadores bitwise

- ❑ Realizan operaciones bit a bit.
- ❑ Se realizan en tiempo constante.



Operadores bitwise

a	b	~ a (not)	a & b (and)	a b (or)	a ^ b (xor)
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Operadores bitwise

Sea $a = 1010$ y $b = 1100$

- $a \& b = 1000$
- $a | b = 1110$
- $a \wedge b = 0110$
- $\sim a = 0101$

En lugar de interpretar sus operandos como V o F (como los operadores booleanos), éstos operan sobre cada uno de sus bits.

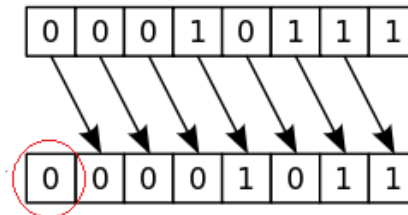


Operadores bitwise

Right shift ($x \gg i$)

- Todos los bits de x corren i posiciones a la derecha y los nuevos bits son llenados con el bit del signo.
- Es equivalente al piso (floor) de la división de x entre 2^i .

Sea $x = 23$
 $x \gg 1 = 00001011 = 11$

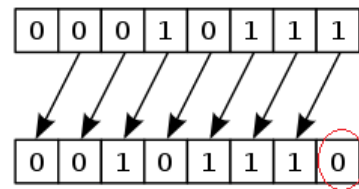


Operadores bitwise

Left Shift ($x \ll i$)




- Todos los bits de x corren i posiciones a la izquierda y los nuevos bits son llenados con ceros.
- Es equivalente al producto de x por 2^i .

Sea $x = 23$
 $x \ll 1 = 00101110 = 46$



Subconjuntos

¿Cómo representamos un subconjunto de un conjunto?

- Arreglo de booleanos (memoria extra y operaciones costosas). 
- Bitset (si requerimos guardar muchos elementos) 
- Máscara de bits (si la cantidad de elementos ≤ 64 , le damos prioridad porque nos servirá más adelante). 

Máscara de bits

- ❑ Podemos usar un entero para representar subconjuntos de un conjunto de hasta 32 elementos (o 64 si usamos long long).
- ❑ El i -ésimo bit del entero (máscara) será 1 si el i -ésimo elemento del conjunto está presente y será 0 si está ausente.
- ❑ Las operaciones relacionadas a los subconjuntos serán bitwise, lo que lo hace eficiente.



Máscara de bits

Supongamos que tenemos el conjunto {6, 3, 8}:

Subconjunto	Máscara de bits	Entero
{6}	001	1
{6, 8}	101	5
{6, 3, 8}	111	7



Tareas frecuentes

- Unión de conjuntos : $a \mid b$
- Intersección de conjuntos : $a \& b$
- Obtener bit de posición i : $(\text{mask} \gg i) \& 1$
- Encender el bit de posición i : $\text{mask} = \text{mask} \mid (1 \ll i)$
- Apagar el bit de posición i : $\text{mask} = \text{mask} \& (\sim(1 \ll i))$
- Cambiar estado del bit i : $\text{mask} = \text{mask} \wedge (1 \ll i)$
- Cantidad de bits encendidos : `__builtin_popcount` `__builtin_popcountll`
- Cantidad de ceros a la derecha : `__builtin_ctz` `__builtin_ctzll`
- Cantidad de ceros a la izquierda : `__builtin_clz` `__builtin_clzll`

Tareas frecuentes

- **Obtener último bit encendido de un entero**

Para un $x = 14$, el último bit encendido es el bit en posición 1:

1	1	1	0
---	---	---	---

El último bit encendido nos lo da la operación: $x \& -x$

(-) es un operador unario que genera el negativo de un número, por ello:

$$-x = \sim x + 1$$



Tareas frecuentes

Demostración

Sea $x = \overline{a10 \dots 0}$

$$-x = \overline{\sim(a10 \dots 0)} + 1, \quad -x = \overline{(\sim a)01 \dots 1} + 1, \quad -x = \overline{(\sim a)10 \dots 0}$$

Finalmente,

$$x \& -x = \overline{a10 \dots 0} \& \overline{(\sim a)10 \dots 0} = \mathbf{0 \dots 010 \dots 0}$$

Obtenemos el bit como potencia de 2.



Problemas

[HackerRank – Lonely Integer](#)

[HackerEarth – Sherlock and XOR](#)

[HackerEarth – Aaryan, Subsequences And Great XOR](#)

[HackerRank – Sum vs Xor](#)

[Codeforces – Bits](#)

Referencias

- ❑ HackerEarth - [Bit Manipulation](#)
- ❑ Topcoder - [A Bit of Fun: Fun with Bits](#)
- ❑ Steven Halim & Felix Halim - **Competitive Programming 3**

¡ Good luck and have fun !