Manipulación de Bits

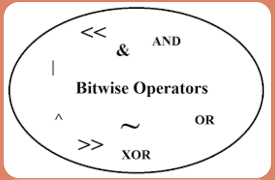
Bach. Rodolfo Mercado Gonzales
Programación Competitiva UPC

Manipulación de Bits

- La mayoría de mejoras en un programa son de “alto nivel” (se centran en la complejidad).

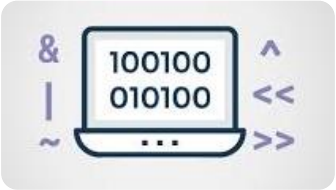
- La manipulación de bits es una de las mejoras a “bajo nivel” más conocidas, la cual permite que nuestro programa sea algo más rápido, simplifica nuestro código e incluso podría mejorar la complejidad de nuestro algoritmo.

Operadores bitwise



Operadores bitwise

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



Operadores bitwise

And $(x \,\&\, y)$

$$\begin{array}{rl} & 10110\ (22) \\ \& & 11010\ (26) \\ \hline = & 10010\ (18) \end{array}$$

Or $(x \mid y)$

$$\begin{array}{rl} & 10110\ (22) \\ | & 11010\ (26) \\ \hline = & 11110\ (30) \end{array}$$

en lugar de interpretar sus operandos como V o F, operan a nivel de bits.

Operadores bitwise

Xor (x^y)

- Retorna un número que tiene los bits encendidos en las posiciones donde exista un bit encendido en x o en y, pero no en ambos.

$$\begin{array}{rl} & 10110\ (22) \\ \hat{} & 11010\ (26) \\ \hline = & 01100\ (12) \end{array}$$

Operadores bitwise

Not ($\sim x$)

- Retorna un número donde todos los bits de x han sido invertidos.

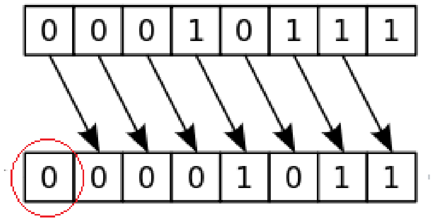
$$
\begin{aligned}
x &= 29 \; 00000000000000000000000000011101 \\
\tilde{}x &= -30 \; 11111111111111111111111111100010
\end{aligned}
$$

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 .

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

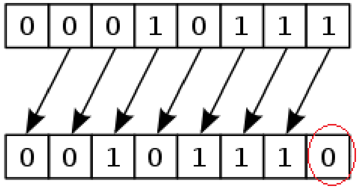


Operadores bitwise

Left Shift ($\boldsymbol{x \ll i}$)

- 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 .

$$\begin{aligned} &Sea\ x = 23 \\ &x \ll 1 \quad = 00101110 \quad = \quad 46 \end{aligned}$$



Propiedades

- Propiedad conmutativa

$a \,\&\, b = b \,\&\, a$

$a \mid b = b \mid a$

$a \wedge b = b \wedge a$

- Propiedad asociativa

$a \,\&\, (b \,\&\, c) = (a \,\&\, b) \,\&\, c$

$a \mid (b \mid c) = (a \mid b) \mid c$

$a \wedge (b \wedge c) = (a \wedge b) \wedge c$

Propiedades

- Elemento neutro

$$a \mid 0 = a \qquad\qquad a \wedge 0 = a$$

- Elemento inverso

$$a \wedge a = 0$$

- Sean los bits $b_1, b_2, \ldots, b_n$

$$b_1 \wedge b_2 \wedge \ldots \wedge b_n = 1 \quad si\ y\ solo\ si\ la\ cantidad\ de\ unos\ es\ impar$$

Propiedades

- Paridad

$$x \,\&\, 1 = 0 \rightarrow x \; es \; par \qquad\qquad x \,\&\, 1 = 1 \rightarrow x \; es \; impar$$

- Complemento a 2

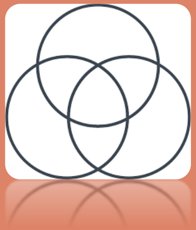
$$-x = \sim x + 1$$

Funciones adicionales

Adicionalmente el compilador g++ provee las siguientes funciones de conteo de bits.

Cantidad de bits encendidos	__builtin_popcount	__builtin_popcountll
Cantidad de ceros al final	__builtin_ctz	__builtin_ctzll
Cantidad de ceros al inicio	__builtin_clz	__builtin_clzll

Representación de subconjuntos



Subconjuntos

Dado un conjunto $\{0, 1, 2, \ldots\ , n-1\}$ ¿Cómo podemos representar sus subconjuntos?

- Arreglo de booleanos (operaciones costosas). ❌
- Máscara de bits (eficiente y de uso frecuente). ✓
- Bitset (eficiente, usado cuando el conjunto es grande). ✓

Máscara de bits

- Podemos usar un entero/long long para representar los subconjuntos de un conjunto de hasta 32/64 elementos.

- El i-ésimo bit de la máscara será 1 si el i-ésimo elemento del conjunto está presente en el subconjunto y será 0 si está ausente.

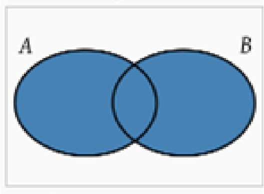
Máscara de bits

Supongamos que tenemos el conjunto $\{6, 3, 8\}$

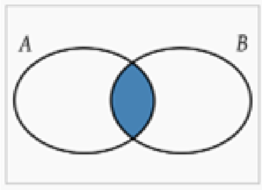
subconjunto	máscara de bits	entero
{6}	001	1
{6, 8}	101	5
{6, 3, 8}	111	7

Tareas frecuentes

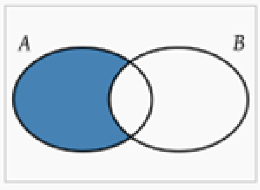
- Unión de subconjuntos $A \mid B$



- Intersección de subconjuntos $A \,\&\, B$

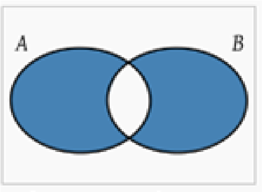


- Diferencia de subconjuntos $A \,\&\, (\sim B)$

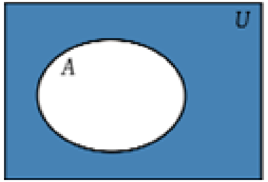


Tareas frecuentes

- Diferencia simétrica $A \wedge B$



- Complemento $\sim A$



Tareas frecuentes

- Obtener el $i - ésimo$ bit $\quad (mask \gg i)\ \&\ 1$
- Encender el $i - ésimo$ bit $\quad mask = mask \mid (1 \ll i)$
- Apagar el $i - ésimo$ bit $\quad mask = mask\ \&\ (\sim(1 \ll i)\ )$
- Invertir el $i - ésimo$ bit $\quad mask\ = mask\ \hat{}\ (1 \ll i)$

Tareas Frecuentes

- Obtener el último bit encendido $x\ \ \&-x$

Demostración

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

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

Finalmente,

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

Tareas frecuentes

- Un número x postivo es potencia de 2 si: $x - (x \,\&\, -x) = 0$

Demostración

Las potencias de 2 solo tienen un bit encendido, entonces si le quitamos el "último bit encendido", debemos obtener 0.

Tareas frecuentes

- Un número x postivo es potencia de 2 si: $x \ \& \ (x - 1) = 0$

Demostración

$x - 1$ tiene los mismos bits de x, excepto el “último bit encendido” y los bits que están a su derecha.
$x \,\&\, (x - 1)$ tiene todos los bits de x, a excepción de su último bit encendido.
Si x es una potencia de 2 solo tendrá un bit encendido, por ende $x \,\&\, (x - 1) = 0$

Bitset

- ❑ Estructura que almacena bits (cada elemento es un 1 o un 0).
- ❑ El i-ésimo elemento será 1 si el i-ésimo elemento del conjunto está presente y será 0 si está ausente.

```
bitset<4> b1; //declaración
bitset<4> b2 ("1100"); //inicialización con string
bitset<4> b3 (5); //inicialización con int
b1[0] = 1; // la posición cuenta desde la derecha
string s = b1.to_string(); // retorna string
int x =  b2.to_ulong(); //retorna int
b1 |= b2; // operaciones bitwise en O(n/32)
b3 >>= 1;
b3 = ~b3;
cout << b3.count(); //cantidad de unos
```

Ejercicios

- [HackerRank – Lonely Integer]
- [HackerEarth – Sherlock and XOR]

Desafíos

- [Codeforces – Little Xor]
- [SPOJ – Happy Coins]
- [SPOJ – Counting Bits]
- [HackerRank – Sum vs XOR]
- [Codechef – The Tom and Jerry Game!]
- [Codeforces – XORwice]
- [Codechef – Chef And Binary Operation]

Referencias

- Antti Laaksonen - Guide to Competitive Programming
- Topcoder – A bit of fun : fun with bits
- HackerEarth – Basic of Bit Manipulation
- Steven Halim & Felix Halim - Competitive Programming 3

"The only way to do great work is to love what you do."

- Steve Jobs