Uso de máscaras

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

08-2022

Contenido

Los algoritmos que hemos analizado en capítulos anteriores se encuentran diseñados para explorar espacios de búsqueda de forma sistemática. No poseen información adicional sobre el problema más allá de la que se proporciona en la definición de este. Todo lo que pueden hacer es generar posibles soluciones parciales y determinar si se ha llegado, o no, al resultado final. A este tipo de búsqueda se les conoce como búsquedas ciegas.

Sin embargo, estos algoritmos pueden ser mejorados usando técnicas de programación y/o estructuras de datos que nos permiten realizar una búsqueda más informada o reducir la memoria empleada en el proceso de generación de la solución. Por ejemplo, las técnicas de Backtracking y Programación Dinámica se ven muy beneficias con el uso de máscaras de bits para almacenar información.

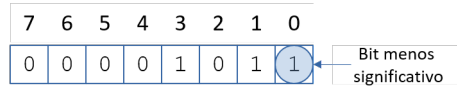
Retomemos el problemas de la suma de subconjuntos:

- Dado un conjunto S de N números enteros positivos, encontrar los subconjuntos que sumen la cantidad C. Si no se encuentra algún subconjunto que cumple, se debe responder con el conjunto vacío.

Ver código "subsetv1.cpp"

Entonces, ¿qué podemos hacer para reducir este tiempo? La respuesta más sencilla sería substituir el método `insert` por alguna operación o función más eficiente. Quizás algo $O(1)$, para que el tiempo de ejecución no se vea tan afectado. Y es aquí donde entran las máscaras de bits. Las máscaras de bits se emplean en la solución de problemas que requieren de tiempo y/o memoria exponencial.

Una máscara de bits no es más que un número binario que representa algo. Tomemos como ejemplo el problema de la suma de subconjuntos. Dado un conjunto inicial, $A = \{x_1, x_2, \cdots, x_n\}$, queremos conocer qué elementos del conjunto suman la cantidad C. Es decir, cuales elementos han sido seleccionados. En este caso, supongamos que el conjunto inicial es $A = 2, 3, 7, 9$. Podemos representar cualquier subconjunto de A usando una máscara de 4 bits. En esta secuencia de bits, el elemento i-ésimo se considera seleccionado si y sólo si el i-ésimo bit de la máscara está prendido, es decir, es igual a 1.



Supongamos queremos representar el subconjunto $X = \{2, 3, 9\}$. Si consideramos que el elemento 1 es el bit menos significativo y así sucesivamente.

Las operaciones básicas que podemos realizar sobre una máscara son establecer (prender un bit), remover (apagar un bit) o verificar (determinar si un bit está prendido). Para realizar estas operaciones usaremos los operadores lógicos or (|), and (&), desplazamiento a la izquierda («) y desplazamiento a la derecha (»).

Digamos que tenemos la máscara del ejemplo anterior, b = 00001011,

- Si queremos establecer el *i*-ésimo bit de una máscara, usaremos la operación: b | (1 « i). Por ejemplo, si queremos establecer el tercer elemento, i = 2:

```
1 << 2 = 00000100
00001011 | 00000100 = 0001111
```

  Así que ahora, el subconjunto también incluye al tercer elemento, $X = \{2, 3, 7, 9\}$.

- Si queremos remover el *i*-ésimo bit de una máscara, debemos usar la siguiente operación: b & !(1 « i). Digamos que ahora queremos remover el segundo elemento, i = 1, del subconjunto:

```
1 << 1 = 00000010
!00000010 = 11111101
00001111 & 11111101 = 00001101
```

  De esta forma, la nueva máscara representará $X = \{2, 7, 9\}$.

- Si queremos verificar si el i-ésimo bit está prendido, usaremos la siguiente operación: b & (1 « i). Si el i-ésimo bit está prendido, el resultado será diferente de cero. Supongamos que queremos verificar si el cuarto elemento se encuentra en el subconjunto, i = 3:

```
      1 << 3 = 00001000
00001101 & 00001000 = 00001000
```

Ya que el resultado es igual a 1, podemos determinar que el cuarto elemento se encuentra en el subconjunto X.

Ver código "subsetv2.cpp"