Programación Dinámica con Máscara de Bits

Por Lautaro Lasorsa - Curso OIA - UNLaM 2020

Operadores de Bits | 1 | Representación

En un programa las variables son almacenadas en memoria, y sus valores son cadenas de bits.

Más específicamente, los **enteros positivos** son almacenados según su representación en binario, así por ejemplo (en enteros de 32 bits):

- 00000000000000000000000000000000 = 0
- 00000000000000000000000000000001 = 1
- 00000000000000000000000000000010 = 2
- 00000000000000000000000000000100 = 4
- 00000000000000000000000000000101 = 5

Operadores de Bits | 2 | Operadores

C++ nos permite trabajar directamente con los bits de esta representación, mediante los operadores & (and, y), | (or, o) y ^(xor), que al utilizarse entre dos enteros realizan la respectiva operación bit a bit.

Ejemplo	0,0	1,0	1,1	2,0	2,1	2,2	3,0	3,1	3,2	3,3
&	0	0	1	0	0	2	0	1	2	3
|	0	1	1	2	3	2	3	3	3	3
^	0	1	0	2	3	0	3	2	1	0

Operadores de Bits | 3 | Shift

Otro operador con el que C++ nos permite manipular los bits es el operador shift.

Concretamente:

- *A >> B* correrá la representación en binario de *A* *B* posiciones a la izquierda, agregando *0* al final. Notar que es equivalente a $\frac{A}{2^B}$
- *A << B* correrá la representación en binario de *A* *B* posiciones a la derecha, agregando *0* al inicio. Notar que es equivalente a $A \cdot 2^B$

Importante: *1<<i* nos genera un entero donde solo el bit en la posición i está encendido.

Máscara de Bits | 1 | Idea

Como cada bit puede tomar valores de verdadero o falso, de forma independiente del resto, puedo usar un entero de 32 bits para representar un vector de booleanos de 32 posiciones. Cuando lo usamos de esta forma, le decimos **máscara de bits.**

Esto es mucho más eficiente en memoria y tiempo, y además podemos usar la máscara como índice de un vector.

En general, nuestras máscaras representan conjuntos, donde cada posición representa un elemento. Lógicamente, un 0 en esa posición indica que el conjunto no contiene dicho elemento, y un 1 que si lo contiene.

Máscara de Bits | 2 | Operaciones 1

Cuando la máscara representa un conjunto, entonces:

- *A* & *B* nos dará la intersección entre *A* y *B*, es decir los elementos que pertenecen a ambos conjuntos.
- *A* | *B* nos dará la unión de *A* y *B*, es decir los elementos que están en al menos uno de los conjuntos
- *A* ^ *B* nos dará la diferencia simétrica entre *A* y *B*, es decir los elementos que están en alguno de ambos conjuntos pero no en el otro.

Notar que las 3 operaciones son conmutativas y asociativas, y además tienen elemento neutro. Además | y & tienen elemento absorbente.

Máscara de Bits | 3 | Operaciones 2

Si el total de elementos es *N*, y *U* es el **conjunto universal** (es decir, decir, que tiene los *N* bits prendidos, contiene a todos los *N* elementos):

- *0* es el elemento neutro de las operaciones | y ^, y *U* es el neutro de &. Es decir: $A \mid 0 = A$; $A \wedge 0 = A$; $A \& U = A$
- *U* es el elemento absorbente de |, y *0* el elemento absorbente de &. Es decir: $A \mid U = U$; $A \& 0 = 0$
- Hacer $A \wedge U$ invierte todos los bits, lo que nos da el complemento de *A*. Es decir, los elementos que no pertenecen al conjunto *A*.

Máscara de Bits | 4 | Generar conjuntos

Notar que utilizando el operador << podemos generar conjuntos de un solo elemento.

Así, *1<<i* nos dará al conjunto que contiene únicamente al elemento i.

Por otro lado, *U = (1<<N) - 1* nos dará el conjunto universal, ya que la representación en binario de $2^N - 1$ tiene los primeros *N* bits encendidos.

Un caso particular es que si tomamos *U ^ (1<<i) = U - (1<<i)* obtendremos un conjunto que contiene a todos los elementos excepto al elemento i.

Máscara de Bits | 5 | Macros

Utilizando lo anterior podemos definir las siguientes macros, que nos servirán para manipular a la máscara A como un vector de booleanos.

- *get(A,i) (bool(A&(1<<(i))))* nos devolverá si *A* contiene al elemento *i*.
- *set(A,i) A |= (1<<(i))* modificará *A* agregando al elemento *i*.
- *delete(A,i) A &= (U-(1<<(i)))* modificará *A* eliminando al elemento *i*.
- *invert(A,i) A ^= (1<<(i))* modificará *A* invirtiendo si contiene o no al elemento *i*.

Además, la función __builtin_popcount(A) nos indicará la cantidad de bits prendidos (cantidad de elementos) en A, aunque no anda en todos los compiladores.

DP con Máscara de Bits | 1 | Introducción

Dado que ahora podemos representar conjuntos utilizando un único entero, y usar ese entero como índice de un vector, podemos plantear una programación dinámica donde una parte del estado de la DP sea un conjunto.

Algunos de los problemas clásicos que pueden resolverse con esta técnica son:

- SubSet Sum en caso general (cuando la suma total puede ser muy grande).
- Coing Change en caso general.
- Travelling Salesman Problem (TSP) / Camino Hamiltoniano.
- Cantidad de cubrimientos con dominos, habiendo fichas ya puestas. (DP con frente)

DP con Máscara de Bits | 2 | SubSet Sum

Este problema consiste en, dado un conjunto S y un valor X, saber si existen elementos de S tales que su suma (o producto, o gcd, o la operación que queramos) sea igual al elemento X.

Cuando todos los valores que se pueden obtener en el proceso son chiquitos, esto puede ser resuelto como se vio en clases anteriores. Sin embargo, si los valores intermedios pueden ser muy grandes y el tamaño de S es pequeño (|S| = N <=20) podemos utilizar esta técnica.

En este caso puntual puede omitirse la DP y simplemente iterar todos los posibles subconjuntos, que son las máscaras de 0 a 2^N-1 en $O(N * 2^N * O(Op))$

DP con Máscara de Bits | 3 | TSP

El Travelling Salesman Problem, o problema del viajante de comercio, consiste en dado un grafo ponderado, encontrar el Ciclo Hamiltoniano (un ciclo simple que pasa por todos los nodos) de costo mínimo. Notar que encontrar un Ciclo Hamiltoniano cualquiera no es más sencillo.

La fuerza bruta, es decir probar todas las permutaciones de los nodos, tiene una complejidad de $O(N!)$. Con esta técnica, podemos hacerlo en $O(N^2 * 2^N)$.

En este caso, nuestro estado tendrá 2 elementos. El conjunto de nodos ya visitados y el último nodo que hemos visitado (en el que estamos parados). La transición será simplemente recorrer cualquier arista válida a un nodo no visitado.

DP con Máscara de Bits | 4 | Cierre

Código de ejemplo SubSet Sum : https://pastebin.com/s5iNJQz5

Código de ejemplo Travelling Salesman Problem: https://pastebin.com/kh5rNxkL

Problemas de ejemplo:

- https://codeforces.com/contest/1051/problem/D
- https://cses.fi/problemset/task/1690

Gracias por ver!