GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: MÁSCARA DE BITS

1. Introducción

En múltiples problemas podemos encontrar que dado un conjunto S de N elementos, se necesita determinar de cuantas formas se puede seleccionar K elementos $(K <= N)$ y que todos los elementos de ese subconjunto cumpla con alguna restricción o condición determinada. O encontrar cuanto subconjunto Ks del conjunto inicial S sus elementos cumple con alguna restricción o condición determinada.

Con la idea algorítmica abordada en esta guía veremos como podemos generar todos los posibles subconjuntos del conjunto inicial **siempre y cuando la cantidad de elementos del conjunto no sobrepase la cantidad de 21.**

2. Conocimientos previos

2.1. Operadores and y desplazamiento hacia la izquierda a nivel de bit

Estos operadores permiten actuar sobre los operandos para modificar un solo bit, se aplican a variables del tipo char , short , int y long y no pueden ser usados con float ó double, sólo se pueden aplicar a expresiones enteras. En ocasiones los operadores de bits se utilizan para compactar la información, logrando que un tipo básico (por ejemplo un long) almacene magnitudes más pequeñas mediante aprovechamientos parciales de los bits disponibles.

2.1.1. AND lógico & (palabra clave bitand)

Este operador binario compara ambos operandos bit a bit, y como resultado devuelve un valor construido de tal forma, que cada bits es 1 si los bits correspondientes de los operandos están a 1. En caso contrario, el bit es 0.

Sintaxis:

$AND - expresion\&equality - expresion$

Ejemplo:

```
int x = 10, y = 20;   // x en binario es 01010, y en binario 10100

int z = x & y;          // equivale a: int z = x bitand y;  z toma el valor
    00000 es decir 0
```

2.1.2. Desplazamiento a izquierda $<<$

Este operador binario realiza un desplazamiento de bits a la izquierda. El bit más significativo (más a la izquierda) se pierde, y se le asigna un 0 al menos significativo (el de la derecha). El operando derecho indica el número de desplazamientos que se realizarán. Los desplazamientos no son rotaciones; los bits que salen por la izquierda se pierden, los que entran por la derecha se rellenan con ceros. Este tipo de desplazamientos se denominan lógicos en contraposición a los cíclicos o rotacionales.

La técnica de la máscara de bit es muy útil cuando se desea generar todos los subconjuntos derivados de un conjunto inicial, tener en cuenta que dicha técnica es solo aplicable cuando el número de elementos del conjunto base no supera los 20, se puede aplicar hasta 21 en dependencia de la cantidad de tiempo límite del problema. Otro aspecto a tener en cuenta en cuanto a esta técnica es que cuando en el conjunto inicial existe elemento repetidos la técnica generará subconjuntos idénticos.

Sintaxis:

$expr - desplazada << expr - desplazamiento$

El patrón de bits de expr-desplazada sufre un desplazamiento izquierda del valor indicado por la expr-desplazamiento. Ambos operandos deben ser números enteros o enumeraciones. En caso contrario, el compilador realiza una conversión automática de tipo. El resultado es del tipo del primer operando. expr-desplazamiento, una vez promovido a entero, debe ser un entero positivo y menor que la longitud del primer operando. En caso contrario el resultado es indefinido (depende de la implementación).

Ejemplo:

```
unsigned long x = 10; // En binario el 10 es 1010

int y = 2;

unsigned long z = x << y;
/* z tienen el valor 40 que en binario es 101000 vea que se adiciono dos ceros
     al final que son los desplazamientos.*/
```

2.2. Conjunto potencia

Se denomina conjunto potencia del conjunto S a al conjunto de todos los subconjuntos de S, ejemplo:

Sea S= {a, b, c} un conjunto conformado por los elementos a, b y c el conjunto potencia de S es {{a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c}, {Ø}}

Por anterior expuesto se puede decir que la cantidad de subconjuntos de un conjunto va ser 2 a la potencia de la cardinalidad del conjunto teniendo como cardinalidad de un conjunto la cantidad de elementos de este. En este caso el conjunto S tiene cardinalidad 3 y la cantidad de subconjuntos es 8.

3. Desarrollo

La técnica de la máscara de bit se encarga de combinar los dos temas abordados en esta presentación anteriormente para eso tomemos como caso de estudio tomemos el conjunto S= {a, b, c} del cual conformaremos todos los posibles subconjunto aplicando la técnica máscara de bit.

Bien como sabemos el conjunto S tiene 8 subconjuntos, pues bien vamos a construir una pequeña tabla con tres columnas en la primera columna estarán todos los números desde el 0 hasta el 7, en la segunda la representación binaria del número a la izquierda mientras en la tercera estarán los elementos del conjunto S que su posición en la representación binario su bit este activo (valor 1). Recuerde que en binario el bit más a la derecha es el primero.

Número	Binario	Elementos seleccionados
0	000	
1	001	a
2	010	b
3	011	a,b
4	100	c
5	101	a,c
6	110	b,c
7	111	a,b,c

La idea de la técnica es iterar desde 0 hasta $2^n - 1$ donde n es la cantidad de elementos del conjunto inicial y para cada número ver cuáles son sus bit activos y seleccionar del conjunto aquellos elementos que sus posiciones se corresponde con las posiciones del bits activos. Ahora duda que puede surgir es como sé que bit del número está activo, sencillo a priori sabemos que el número va estar conformado por n bits donde n es cantidad de elementos del conjunto pues haremos una operación and entre el número y 1 desplazado hacia la izquierda x lugares donde x va ser un número entre 0 y n.

4. Implementación

4.1. C++

```
vector<char> s('a','b','c');
int n= s.size();
for (int i=0; i< (1<<n); i++)
{
        int number=i;
        vector<char> k;
        for (int e=0; e< n; e++)
        {
                if( number & (1<< e) )
                {
                        k.push_back(s[e]);
                }
        }
        /*Ya  k tiene los elementos que conforma el subconjunto i-nesimo*/
        cout<< "El sub-conjunto "<<e+1<<"es:"<<endl;

        cout<<"{ ";
        for(int h=0; h<k.size(); h++)
            cout<<k[h]<<" ";
```

```
        cout<<"}"<<endl;
    }
```

4.2. Java

```
    char [] s= {'a','b','c'};
    int n= s.length;
    for (int i=0; i< (1<<n); i++)
    {
        int number=i;
        ArrayList<Character> k =new ArrayList<Character>() ;
        for (int e=0; e< n; e++)
        {
            if( (number & (1<< e))!=0 )
            {
                k.add(s[e]);
            }
        }
        /*Ya  k tiene los elementos que conforma el subconjunto i-nesimo*/
        System.out.println("El sub-conjunto "+(e+1)+"es:");

        System.out.print("{ ");
        for(int h=0; h<k.size(); h++)
            System.out.print(k.get(h)+" ");
        System.out.println("}");
    }
```

5. Complejidad

Analizando la complejidad del algoritmo nos damos cuenta que el mismo es $2^n * n$ donde n como ya se ha reiterado en otras ocasiones es la cantidad de elementos del conjunto inicial. Por lo que dicha técnica es aplicable para n menor igual que 20 ya 21 habría que analizar el tiempo límite del problema así como las adecuaciones hechas dentro de este algoritmo incluso con 20 elementos hay que hilar una óptima solución.

Para n igual 20 el número de iteraciones es 20971520 mientras para 21 es 44040192.

6. Aplicaciones

Los números primos son utilizados en diferentes tipos de problemas o ejercicios, en algunos son la parte fundamental de a solución porque el problema trata propiamente de ellos, en otros son las parte de la base o sireven de apoyo para elaborar una solución como puede ser los problemas de hashing, factorización, cantidad de divisores, inverso multiplicativo.

7. Ejercicios propuestos

A continuación una lista de ejercicios que se resuelven utilizando esta técnica:

- CodeForce - Preparing Olympiad.
- CodeForce - Petr and a Combination Lock
- MatCom Grader - B - Bolsa de Caramelos. Se debe optimizar un poco para que entre en tiempo Si el subcojunto que esta construyendo supera en la k o la s no debe seguir construyendo ese subconjunto.