Complete Search y Bitmask

Víctor Racsó Galván Oyola

12 de mayo de 2021

Índice

Complete Search

Concepto

Idea

Trata de buscar todas las posibilidades antes de dar una respuesta final.
Ejemplo: Cuando buscas algo en una cómoda y tienes que abrir todos los cajones.

Concepto

Idea

Trata de buscar todas las posibilidades antes de dar una respuesta final.
Ejemplo: Cuando buscas algo en una cómoda y tienes que abrir todos los cajones.

Ventajas

Si la solución es implementada adecuadamente, la respuesta dada siempre es correcta, por lo que puede servir como referencia para hacer pruebas de algunas soluciones eficientes pero incorrectas.

Concepto

Idea

Trata de buscar todas las posibilidades antes de dar una respuesta final.
Ejemplo: Cuando buscas algo en una cómoda y tienes que abrir todos los cajones.

Ventajas

Si la solución es implementada adecuadamente, la respuesta dada siempre es correcta, por lo que puede servir como referencia para hacer pruebas de algunas soluciones eficientes pero incorrectas.

Desventajas

Puede demorar demasiado, la mayoría de veces su eficiencia no es la suficiente para problemas de nivel intermedio o alto.

Problema de muestra

¿Cómo verificamos si un número es primo?

Dado un entero $n \geqslant 2$, determinar si este es primo o es compuesto.

Problema de muestra

¿Cómo verificamos si un número es primo?

Dado un entero $n \geqslant 2$, determinar si este es primo o es compuesto.

Solución I

Probar si algún entero en el rango $[2, n-1]$ divide a n.
Complejidad: $O(n)$

Problema de muestra

¿Cómo verificamos si un número es primo?

Dado un entero $n \geqslant 2$, determinar si este es primo o es compuesto.

Solución I

Probar si algún entero en el rango $[2, n-1]$ divide a n.
Complejidad: $O(n)$

Solución II

Notando que si n **no** es primo, en el peor de los casos será de la forma $n = p^2$ para algún primo p, así que nos basta analizar los enteros en el rango $[2, \lfloor\sqrt{n}\rfloor]$.
Complejidad: $O(\sqrt{n})$.

Problema de muestra

¿Cómo verificamos si un número es primo?

Dado un entero $n \geqslant 2$, determinar si este es primo o es compuesto.

Solución I

Probar si algún entero en el rango $[2, n-1]$ divide a n.
Complejidad: $O(n)$

Solución II

Notando que si n **no** es primo, en el peor de los casos será de la forma $n = p^2$ para algún primo p, así que nos basta analizar los enteros en el rango $[2, \lfloor\sqrt{n}\rfloor]$.
Complejidad: $O(\sqrt{n})$.

Bonus: ¿Y si hay múltiples n en un rango pequeño?

Si ahora nos dan $T \leqslant 10^5$ enteros $n_i \leqslant 10^8$, ¿Qué hacemos? ¿Y si $n_i \leqslant 10^{18}$?

Problema de muestra - II

¿Cómo contamos la cantidad de divisores de un entero n?

Dado n, hallar la cantidad de divisores positivos de n.

Problema de muestra - II

¿Cómo contamos la cantidad de divisores de un entero n?

Dado n, hallar la cantidad de divisores positivos de n.

Solución I

Verificar todos los enteros en $[1, n]$.
Complejidad: $O(n)$

Problema de muestra - II

¿Cómo contamos la cantidad de divisores de un entero n?

Dado n, hallar la cantidad de divisores positivos de n.

Solución I

Verificar todos los enteros en $[1, n]$.
Complejidad: $O(n)$

Solución II

Notando que si tenemos un divisor $d \leqslant \sqrt{n}$, este genera otro divisor d' de n tal que $d' \geqslant \sqrt{n}$, podemos simplemente verificar todos los enteros en $[1, \sqrt{n}]$ para hacer el conteo.
Complejidad: $O(\sqrt{n})$.

Problema de muestra - II

¿Cómo contamos la cantidad de divisores de un entero n?

Dado n, hallar la cantidad de divisores positivos de n.

Solución I

Verificar todos los enteros en $[1, n]$.
Complejidad: $O(n)$

Solución II

Notando que si tenemos un divisor $d \leqslant \sqrt{n}$, este genera otro divisor d' de n tal que $d' \geqslant \sqrt{n}$, podemos simplemente verificar todos los enteros en $[1, \sqrt{n}]$ para hacer el conteo.
Complejidad: $O(\sqrt{n})$.

Solución III

Hallando la descomposición en factores primos de n podemos obtener la respuesta multiplicando los exponentes aumentados en 1.
Complejidad: $O(\text{mín}\{\sqrt{n}, mpf(n)\})$, donde $mpf(x)$ es el máximo factor primo de x.

Intentemos un problema en Codeforces

Cut Ribbon

Dada la longitud n de una cinta, determinar la máxima cantidad de piezas que se pueden obtener de ella haciendo cortes de longitud a, b o c.

Intentemos un problema en Codeforces

Cut Ribbon

Dada la longitud n de una cinta, determinar la máxima cantidad de piezas que se pueden obtener de ella haciendo cortes de longitud a, b o c.

Formalizando el problema

Dados n, a, b, c, definir x, y y z enteros no negativos tales que:

$$ax + by + cz = n$$

Y además $(x + y + z)$ sea el máximo posible.

Intentemos un problema en Codeforces

Cut Ribbon

Dada la longitud n de una cinta, determinar la máxima cantidad de piezas que se pueden obtener de ella haciendo cortes de longitud a, b o c.

Formalizando el problema

Dados n, a, b, c, definir x, y y z enteros no negativos tales que:

$$ax + by + cz = n$$

Y además $(x + y + z)$ sea el máximo posible.

Solución I

Probamos todas las tuplas posibles (x, y, z) con $x, y, z \leqslant n$.
Complejidad: $O(n^3)$. Veredicto: TLE

Intentemos un problema en Codeforces - II

Solución II

Si fijamos el valor de x y y, entonces $z = \frac{n-ax-by}{c}$ obligatoriamente. Ya que z debe ser entero, debemos asegurarnos de que $n - ax - by$ sea múltiplo de c.
Complejidad: $O(n^2)$. Veredicto: AC

Intentemos un problema en Codeforces - II

Solución II

Si fijamos el valor de x y y, entonces $z = \frac{n-ax-by}{c}$ obligatoriamente. Ya que z debe ser entero, debemos asegurarnos de que $n - ax - by$ sea múltiplo de c.
Complejidad: $O(n^2)$. Veredicto: AC

Código en C++

```
int ans = 0;
for(int x = 0; a * x <= n; x++){
        for(int y = 0; a * x + b * y <= n; y++){
                if((n - a * x - b * y) % c) continue;
                ans = max(ans, x + y + (n - a * x - b * y) / c);
        }
}
printf("%d\n", ans);
```

Bitmask

Complemento a 2

La representación de enteros en una computadora para una variable de b bits toma 1 bit para representar el signo (0 si es positivo y 1 si es negativo) y $b-1$ bits para representar el "valor" del número.

Complemento a 2

La representación de enteros en una computadora para una variable de b bits toma 1 bit para representar el signo (0 si es positivo y 1 si es negativo) y $b-1$ bits para representar el "valor" del número.

Enteros positivos

Su bit de signo es 0.
Sus $b-1$ bits de "valor" equivalen a la representación binaria del entero.

Complemento a 2

La representación de enteros en una computadora para una variable de b bits toma 1 bit para representar el signo (0 si es positivo y 1 si es negativo) y $b-1$ bits para representar el "valor" del número.

Enteros positivos

Su bit de signo es 0.
Sus $b-1$ bits de "valor" equivalen a la representación binaria del entero.

Enteros negativos

Su bit de signo es 1.
Sus $b-1$ bits de "valor" se halla usando la siguiente expresión:

$$Valor(-x) = Inv(Valor(x-1))$$

Donde *Inv* es una función que invierte los bits (cambia los 0s por 1s y viceversa) de la representación binaria de un número.

Complemento a 2

La representación de enteros en una computadora para una variable de b bits toma 1 bit para representar el signo (0 si es positivo y 1 si es negativo) y $b-1$ bits para representar el "valor" del número.

Enteros positivos

Su bit de signo es 0.
Sus $b-1$ bits de "valor" equivalen a la representación binaria del entero.

Enteros negativos

Su bit de signo es 1.
Sus $b-1$ bits de "valor" se halla usando la siguiente expresión:

$$Valor(-x) = Inv(Valor(x-1))$$

Donde *Inv* es una función que invierte los bits (cambia los 0s por 1s y viceversa) de la representación binaria de un número.

Rango efectivo

Ya que $-x$ depende de $x-1$, podremos representar hasta el -2^b en negativos, mientras que los positivos solo hasta el 2^b-1, dándonos un rango de $[-2^b, 2^b-1]$.

Desplazamiento de bits (<< / >>)

Podemos mover la representación binaria de un número hacia la izquierda o derecha, considerando que no existen órdenes negativos en la misma.

- Izquierda (`x << k`): Es el equivalente a multiplicar x por 2^k. `3 << 2 = 12`
- Derecha (`x >> k`): Es el equivalente a asignar $\left\lfloor \frac{x}{2^k} \right\rfloor$ a x. `3 >> 2 = 0`.

Desplazamiento de bits (<< / >>)

Podemos mover la representación binaria de un número hacia la izquierda o derecha, considerando que no existen órdenes negativos en la misma.

- Izquierda (`x << k`): Es el equivalente a multiplicar x por 2^k. `3 << 2 = 12`
- Derecha (`x >> k`): Es el equivalente a asignar $\left\lfloor \frac{x}{2^k} \right\rfloor$ a x. `3 >> 2 = 0`.

And (Y lógico) &

Aplica la operación Y lógico (el resultado es 1 si ambos argumentos son 1, en caso contrario es 0) a cada orden de bits.
`3 & 7 = 3`, pues:

$$\begin{aligned} \text{Orden } 2: \quad & 0 \quad \& \quad 1 \quad = 0 \\ \text{Orden } 1: \quad & 1 \quad \& \quad 1 \quad = 1 \\ \text{Orden } 0: \quad & 1 \quad \& \quad 1 \quad = 1 \end{aligned}$$

Or (O lógico) |

Aplica la operación O lógico (el resultado es 1 si alguno de los argumentos es 1, en caso contrario es 0) a cada orden de bits.
2 | 6 = 6, pues:

$$\begin{array}{lccc} \text{Orden } 2: & 0 & | & 1 \quad = 1 \\ \text{Orden } 1: & 1 & | & 1 \quad = 1 \\ \text{Orden } 0: & 0 & | & 0 \quad = 0 \end{array}$$

Or (O lógico) |

Aplica la operación O lógico (el resultado es 1 si alguno de los argumentos es 1, en caso contrario es 0) a cada orden de bits.
2 | 6 = 6, pues:

$$\begin{aligned} \text{Orden } 2: \quad & 0 \mid 1 = 1 \\ \text{Orden } 1: \quad & 1 \mid 1 = 1 \\ \text{Orden } 0: \quad & 0 \mid 0 = 0 \end{aligned}$$

Xor (O exclusivo lógico) $\oplus$

Aplica la operación O exclusivo lógico (el resultado es 1 si los argumentos son diferentes, en caso contrario es 0) a cada orden de bits.
2 $\oplus$ 6 = 4, pues:

$$\begin{aligned} \text{Orden } 2: \quad & 0 \oplus 1 = 1 \\ \text{Orden } 1: \quad & 1 \oplus 1 = 0 \\ \text{Orden } 0: \quad & 0 \oplus 0 = 0 \end{aligned}$$

En la mayoría de lenguajes de programación se usa el operador ^

Representando un conjunto como un entero

Podemos aprovechar la representación binaria de un entero para representar conjuntos, de manera que si tenemos una secuencia a de longitud n, entonces el i-ésimo (indexado desde 0) bit de mi máscara b estará prendido si el conjunto que representa b contiene al elemento a_i.

Representando un conjunto como un entero

Podemos aprovechar la representación binaria de un entero para representar conjuntos, de manera que si tenemos una secuencia a de longitud n, entonces el i-ésimo (indexado desde 0) bit de mi máscara b estará prendido si el conjunto que representa b contiene al elemento a_i.

Ejemplo

Sea $a = (1, 4, 9, 16, 25)$ y $b = 5$, entonces:

$$b = 00101_{(2)} \rightarrow S = \{1, 9\}$$

Representando un conjunto como un entero

Podemos aprovechar la representación binaria de un entero para representar conjuntos, de manera que si tenemos una secuencia a de longitud n, entonces el i-ésimo (indexado desde 0) bit de mi máscara b estará prendido si el conjunto que representa b contiene al elemento a_i.

Ejemplo

Sea $a = (1, 4, 9, 16, 25)$ y $b = 5$, entonces:

$$b = 00101_{(2)} \rightarrow S = \{1, 9\}$$

Pregunta

Si a y b representan conjuntos sobre la misma secuencia s, ¿Qué representan las operaciones *And*, *Or* y *Xor*?

Representando un conjunto como un entero

Podemos aprovechar la representación binaria de un entero para representar conjuntos, de manera que si tenemos una secuencia a de longitud n, entonces el i-ésimo (indexado desde 0) bit de mi máscara b estará prendido si el conjunto que representa b contiene al elemento a_i.

Ejemplo

Sea $a = (1, 4, 9, 16, 25)$ y $b = 5$, entonces:

$$b = 00101_{(2)} \rightarrow S = \{1, 9\}$$

Pregunta

Si a y b representan conjuntos sobre la misma secuencia s, ¿Qué representan las operaciones *And*, *Or* y *Xor*?

Nota

Una máscara también puede representar uno de 2 estados posibles para cada elemento, el usual es inclusión pero también hay otras formas de aprovecharlo.

Intentemos un problema en Codeforces

Dreamoon and WiFi

Dada una secuencia a de + y −, los cuales significan agregar 1 o -1 a una variable inicializada en 0, se nos da otra secuencia b compuesta por +, − y ?. Se nos pide hallar la cantidad de formas en las que podemos asignar un signo a los ? de b y que la variable tenga el mismo valor final en ambas secuencias.

Intentemos un problema en Codeforces

Dreamoon and WiFi

Dada una secuencia a de + y −, los cuales significan agregar 1 o -1 a una variable inicializada en 0, se nos da otra secuencia b compuesta por +, − y ?. Se nos pide hallar la cantidad de formas en las que podemos asignar un signo a los ? de b y que la variable tenga el mismo valor final en ambas secuencias.

Solución

Guardamos todos los ? en un arreglo v y consideramos todas las posibles máscaras de longitud $|v|$ de manera que si el i-ésimo bit está prendido, al i-ésimo ? será asignado el signo + y, en caso contrario, −.
Complejidad: $O(n \cdot 2^n)$

Intentemos un problema en Codeforces

Dreamoon and WiFi

Dada una secuencia a de + y −, los cuales significan agregar 1 o -1 a una variable inicializada en 0, se nos da otra secuencia b compuesta por +, − y ?. Se nos pide hallar la cantidad de formas en las que podemos asignar un signo a los ? de b y que la variable tenga el mismo valor final en ambas secuencias.

Solución

Guardamos todos los ? en un arreglo v y consideramos todas las posibles máscaras de longitud $|v|$ de manera que si el i-ésimo bit está prendido, al i-ésimo ? será asignado el signo + y, en caso contrario, −.
Complejidad: $O(n \cdot 2^n)$

Bonus

Determine cómo iterar sobre todos los subconjuntos de una máscara b en $O(2^n)$ (diseñar el `for`, no el análisis de los estados de cada elemento). Luego, calcule la complejidad de iterar sobre todos los subconjuntos de todos los enteros entre 0 y $2^n - 1$ usando su `for` anterior.