

Escritura del problema del Binario Inverso

María José Niño Rodríguez¹

David Santiago Quintana Echavarria¹

¹Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
{ma.nino, quintanae-david}@javeriana.edu.co

16 de agosto de 2022

Resumen

En este documento se presenta la formalización del problema del cálculo del binario inverso, junto con la descripción de dos algoritmos que lo solucionan, uno de forma iterativa y otro con la estrategia de dividir y vencer. Además, se presenta el análisis de la complejidad de esos dos algoritmos. **Palabras clave:** representación binaria, representación binaria inversa, algoritmo, dividir y vencer, iterativo, formalización, complejidad.

Índice

1. Introducción	1
2. Formalización del problema	1
2.1. Definición del problema del “binario inverso”	2
3. Algoritmos de solución	2
3.1. Binario Inverso dividir y vencer	2
3.1.1. Análisis de complejidad	3
3.1.2. Invariante	3
3.2. Binario Inverso Iterativo	4
3.2.1. Análisis de complejidad	4
3.2.2. Invariante	4

1. Introducción

Los algoritmos iterativos y de dividir y vencer ofrecen la posibilidad de tener diferentes formas de abarcar un mismo problema y elegir cuál es mejor según el caso. En este documento se exponen los dos para el cálculo de la representación binaria inversa de un número, con el objetivo de mostrar: la formalización del problema (sección 2), la escritura formal de cada algoritmo (sección 3) junto con su análisis de complejidad e invariante.

2. Formalización del problema

A partir de un número natural (\mathbb{N}), obtener su representación binaria y posteriormente calcular su representación binaria inversa.

2.1. Definición del problema del “binario inverso”

Así, el problema del binario inverso se define a partir de:

1. un número natural (\mathbb{N}) y
2. la inversión de la representación binaria del número natural (\mathbb{N})

producir un nuevo número natural.

■ Entradas:

- $n \in \mathbb{N}$.

■ Salidas:

- $n' \in \mathbb{N}$.

3. Algoritmos de solución

3.1. Binario Inverso dividir y vencer

La idea de este algoritmo es: dividir el número natural en dos conjuntos, cada uno con la mitad de los bits del numero, dividiendo cada subconjunto consecutivamente hasta llegar a un conjunto de un solo bit, retornando su valor natural en representación binaria inversa.

Algoritmo 1 Binario Inverso Dividir y Vencer

Require: $n \in \mathbb{N}$

Ensure: El resultado será un $n' \in \mathbb{N}$ que es el número de su representación binaria inversa.

```
1: procedure INVERSOBINARIO( $n$ )
2:   if  $n = 0$  then
3:     return 0
4:   end if
5:   return INVERSOBINARIOAUXILIAR( $n, 0, \lfloor \log_2(n) \rfloor + 1, \lfloor \log_2(n) \rfloor + 1$ )
6: end procedure
```

Algoritmo 2 Binario Inverso Dividir y Vencer Auxiliar

Require: $n, b, e, t \in \mathbb{N}$

Ensure: El resultado será un $n' \in \mathbb{N}$ que es el número de su representación binaria inversa.

```
1: procedure INVERSOBINARIOAUXILIAR( $n, b, e, t$ )
2:   if  $b = e$  then
3:     if  $n = 0$  then
4:       return 0
5:     else
6:       return  $2^{t-e-1}$ 
7:     end if
8:   end if
9:    $q \leftarrow \lfloor (b + e) / 2 \rfloor$ 
10:   $n1 \leftarrow \text{MASCARA}(n, b, q)$ 
11:   $n2 \leftarrow \text{MASCARA}(n, q+1, e)$ 
12:  return INVERSOBINARIOAUXILIAR( $n1, b, q, t$ ) + INVERSOBINARIOAUXILIAR( $n2, q + 1, e, t$ )
13: end procedure
```

Algoritmo 3 Mascara

Require: $n, b, e \in \mathbb{N}$ **Ensure:** El resultado será un $n' \in \mathbb{N}$ que tiene prendidos los bits desde el bit b hasta el e del n .

```
1: procedure MASCARA( $n, b, e$ )
2:    $mask \leftarrow 2^{(e-b+1)} - 1$ 
3:    $mask \leftarrow mask \ll b$ 
4:    $mask \leftarrow nmask$ 
5:   return  $mask$ 
6: end procedure
```

3.1.1. Análisis de complejidad

En el caso del algoritmo de dividir y vencer la complejidad se obtuvo por medio del teorema maestro

$$f(x) = \begin{cases} O(1) & ; \text{ num} = 0 \\ 2T(\frac{n}{2}) + O(1) & ; \text{ num} > 0 \end{cases}$$

Posteriormente se evalúa el primer caso para verificar que cumpla

$$\begin{aligned} f(n) &\in O(n^{\log_b(a-\varepsilon)})n^{\log_b(a-\varepsilon)} = 1 \\ n^{\log_2(2-\varepsilon)} &= 1 \\ \varepsilon &= 1 \\ n^{\log_2(2-1)} &= 1 \\ n^{\log_2(1)} &= 1 \\ n^0 &= 1 \end{aligned}$$

Como el primer caso cumple se obtiene que la complejidad es

$$\begin{aligned} T(n) &\in \Theta(n^{\log_a(b)}) \\ &\Theta(n^{\log_2(2)}) \\ &\Theta(n^1) \\ T(n) &= \Theta(n) \end{aligned}$$

3.1.2. Invariante

El número en cada iteración es dividido por la mitad de sus bits, se envía de manera recursiva sus primeros bits calculando la sumatoria de estos, de misma manera se adiciona el calculo con los bits restantes. Sumando sus valores representativos a partir del índice.

1. Inicio: n , se mantiene el número original.
2. Iteración: n , por medio de mascaras se calcula en $n1$ la representación de la primera mitad de los bits del número y en $n2$ los bits restantes. Se retorna la suma de la llamada de la función, pasando $n1$ y $n2$ con sus respectivos pivotes. Así consecutivamente de manera recursiva hasta que los pivotes se encuentren en la misma posición, donde se retorna la representación del bit en ese índice de manera inversa.
3. Terminación: Retorna el valor en entero de la representación binaria inversa de la entrada.

3.2. Binario Inverso Iterativo

La idea de este algoritmo es: sumar la potencia de 2 elevado a la posición del bit correspondiente y multiplicarlo por el modulo 2 del número, posteriormente el número es dividido entre 2. El procedimiento anterior se repite hasta que el número sea igual a 0

Algoritmo 4 Binario Inverso Iterativo

Require: $n \in \mathbb{N}$

Ensure: El resultado será un $n' \in \mathbb{N}$ que es el número de su representación binaria inversa.

```
1: procedure INVERSOBINARIOITERATIVO( $n$ )
2:    $tam \leftarrow \lfloor \log_2(n) \rfloor$ 
3:    $res \leftarrow 0$ 
4:   while  $n \neq 0$  do
5:      $res \leftarrow res + (2^{tam} * (n \% 2))$ 
6:      $n \leftarrow n / 2$ 
7:      $tam \leftarrow tam - 1$ 
8:   end while
9:   return  $res$ 
10: end procedure
```

3.2.1. Análisis de complejidad

Por inspección de código: hay un ciclo, entonces el algoritmo es de complejidad $O(n)$.

3.2.2. Invariante

Después de cada iteración controlada por n , el valor de n vale la mitad y en el acumulador res se almacena la sumatoria de las potencias de 2.

1. Inicio: $res = 0$, el acumulador vale 0.
2. Iteración: $n \neq 0$, se aumenta en el acumulador 2 potenciado el índice del bit correspondiente, y el valor del numero vale la mitad en cada iteración.
3. Terminación: $n = 0$, se retorna el valor en representación \mathbb{N} del binario inverso de la entrada.