

Escribir Un Algoritmo Para Encontrar Un Elemento En Una Secuencia*

Juan Pablo Gonzalez Rincon¹, Matias Felipe Gonzalez Valencia.¹, Laura Isabel Montero Blanco¹

^a*Pontificia Universidad Javeriana, Bogota, Colombia*

Abstract

Este documento presenta un algoritmo basado en "dividir y vencer" para calcular la representación booleana de un número natural. El algoritmo descompone recursivamente el número en sus componentes binarios y combina los resultados para obtener la secuencia final de bits. Se implementa en Python y se analiza su complejidad temporal y espacial.

Keywords: Dividir y vencer, representación booleana, algoritmos recursivos, complejidad algorítmica.

1. Análisis del Problema

Dado un número natural $n \in \mathbb{N}$, se calculará su representación booleana B_n .

Formalmente, la representación booleana de un número n se define como la secuencia de bits que representa a n en el sistema binario, es decir:

$$B_n = \{b_i \mid b_i \in \{0, 1\}, 0 \leq i < k, k = \lceil \log_2(n+1) \rceil\}$$

donde:

- n es un número natural.
- B_n es la representación booleana de n , la cual es un vector de bits.
- k es la longitud de la secuencia booleana necesaria para representar n en binario.
- b_i representa los bits individuales de la secuencia booleana, tales que cada b_i es igual a 0 o 1.

*Este documento presenta la escritura formal de un algoritmo.

Email addresses: `gonzalez_juanp@javeriana.edu.co` (Juan Pablo Gonzalez Rincon),
`matias_gonzalez@javeriana.edu.co` (Matias Felipe Gonzalez Valencia.),
`Limontero@javeriana.edu.co` (Laura Isabel Montero Blanco)

El objetivo es construir la secuencia B_n a partir de n , utilizando operaciones de división y residuo para obtener los bits b_i de manera recursiva, de modo que la representación B_n cumpla con las propiedades del sistema binario para representar a n de forma exacta.

2. Algoritmos de solución

2.1. Algoritmo Recursivo

El algoritmo recursivo descompone un número X en rangos de bits, calculando su representación binaria dentro de un intervalo definido por b y e . Se basa en la técnica de "dividir y vencer", donde X se descompone mediante llamadas recursivas para calcular los bits individuales en el rango especificado. Finalmente, combina los bits obtenidos para formar la secuencia binaria completa.

Algorithm 1 Conversión Binaria Auxiliar con Rango

```

1: procedure CB( $X, b, e$ )
2:   if  $e < b$  then
3:     return []
4:   else
5:      $q \leftarrow (b + e) // 2$ 
6:      $d \leftarrow (X // (2^q)) \% 2$ 
7:      $l \leftarrow \text{CB}(X, b, q - 1)$ 
8:      $h \leftarrow \text{CB}(X, q + 1, e)$ 
9:     return  $h + [d] + l$ 
10:  end if
11: end procedure

```

2.1.1. Invariante

Las sentencias de retorno de la función siempre devolverán la secuencia binaria del número ingresado.

- Inicio: Caso base: Se retorna la secuencia vacía.
- Avance: Cualquier número con una secuencia no vacía: Se hace la transformación a binario y se retorna la secuencia booleana.

2.1.2. Análisis de Complejidad

La complejidad del algoritmo se analiza en términos de la ecuación de recurrencia: $T(n) = T\left(\frac{n}{2}\right) + O(n)$

donde $T(n)$ es el tiempo de ejecución para un número con n posiciones en su traducción booleana y la partición del algoritmo tiene una complejidad de $O(n)$ en el peor de los casos.

2.2. Algoritmo Iterativo

El algoritmo iterativo convierte el número natural n en su representación binaria utilizando un bucle. Calcula los bits uno por uno, almacenándolos en una lista, hasta que el número se reduce a 0.

Algorithm 2 Conversión Iterativa a Binario

```
1: procedure CBI( $n$ )  
2:    $res \leftarrow []$   
3:   while  $n > 0$  do  
4:      $r \leftarrow n \% 2$   
5:      $res \leftarrow [r] + res$   
6:      $n \leftarrow n / 2$   
7:   end while  
8:   if  $res = []$  then  
9:     return  $[0]$   
10:  end if  
11:  return  $res$   
12: end procedure
```
