Taller 2 Dividir y vencer*

Julian Araque, Juliana Lugo, Pablo Gonzalez[a]

[a] *Pontificia Universidad Javeriana, Bogotá, Colombia*

Abstract

En este documento se presenta el análisis, diseño e implementación de un algoritmo recursivo para encontrar la representación booleana de un número natural. Se incluye el análisis de complejidad utilizando el teorema maestro y un análisis de invariante.

Keywords: algoritmo, representación booleana, complejidad, recursividad.

1. Análisis del problema

Sea n un número natural la representación booleana de n es una secuencia de bits (0s y 1s) que indica qué potencias de 2 suman para dar como resultado a n.

2. Diseño del problema

Entrada formal:

- n: Un número natural perteneciente a $\mathbb{N}$.

Salida formal:

- Una lista de bits que representa la representación booleana de n.

Descripción del proceso:

- Divide el número por 2 recursivamente hasta llegar al caso base (0 o 1).
- En cada paso, guarda el residuo de la división por 2.
- Combina los residuos para formar la representación booleana completa.

*Este documento presenta la escritura formal de un algoritmo.

Email address: julugo@javeriana.edu.co, j-araque@javeriana.edu.co, p-gonzalez@javeriana.edu.co (Julian Araque, Juliana Lugo, Pablo Gonzalez)

3. Algoritmo

Algoritmo 1 Representación booleana de un número natural.

```
procedure BOOL_REP(n)
    if n == 0 then
        return []
    else
        return BOOL_REP(n//2) + [n %2]
    end if
end procedure
```

3.1. Implementación en Python

Algoritmo 2 Implementación en Python para la representación booleana.

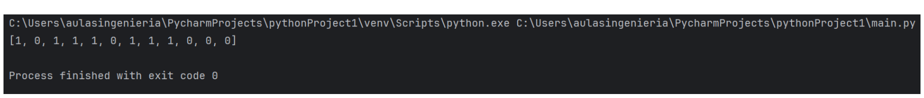
```
def bool_rep(n):
if n == 0 then
    return []
else
    return bool_rep(n // 2) + [n %2]
end if
n = 5
print(bool_rep(n)) # Output: [1, 0, 1]
```

3.2. Comparación Experimental

Algoritmo 3 Comparación Experimental con Temporización.

```
import time
import random
def test_bool_rep():
  start_time = time.time()
for i in range(30000): do
    n = random.randint(0, 10**6)
    bool_rep(n)
end for
  end_time = time.time()
  print("Tiempo total:", end_time - start_time)
test_bool_rep()
```

3.3. Compilación comparación experimental



```
C:\Users\aulasingenieria\PycharmProjects\pythonProject1\venv\Scripts\python.exe C:\Users\aulasingenieria\PycharmProjects\pythonProject1\main.py
[1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0]

Process finished with exit code 0
```

Figura 1: Comparación experimental

4. Análisis de Complejidad

La recurrencia que describe este problema es:

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

Aplicación del Teorema Maestro:

$$\log_b a = \log_2 1 = 0$$

$$Sif(n) \in \Theta(n^{\log_b a} \log n), entoncesT(n) \in \Theta(n^{\log_b a} \log n)$$

$$f(n) = O(1) \in y \in n^{\log_b a} \cdot \log n = \log n)$$

$$T(n) \in \Theta(\log n)$$

5. Análisis de invariante

Inicialmente, n es el número original. Después de la primera llamada recursiva, n se reduce a $\frac{n}{2}$, y se almacena el bit correspondiente a $n\,\%2$. Este proceso se repite hasta que $n = 0$, momento en el cual la secuencia de bits está completa.