Escritura del problema de la representación binaria inversa

David Enrique Palacios García[1] Karen Sofia Coral Godoy[1]

[1]Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
{david_palacios, corallg_ksofia}@javeriana.edu.co

16 de agosto de 2022

Resumen

En este documento se presenta la formalización del problema de la inversión de una cadena, junto con la descripción de dos algoritmos que lo solucionan. Además, se presenta un análisis experimental de la complejidad de esos dos algoritmos. **Palabras clave:** inversión, algoritmo, formalización, experimentación, complejidad, cadena.

Índice

1. Introducción

Invertir una cadena puede llegar a ser útil para encontrar similitudes entre datos, tal así, que existe un gran número de formas para hacerlo. El ejemplo práctico que se tratará en este documento será tomar un número natural N, convertirlo a cadena binaria, invertir esta cadena y, finalmente, obtener el valor entero de la cadena invertida. Para resolver este problema se presentan dos algoritmos que lo solucionan, con el objetivo de mostrar: la formalización del problema (sección 2), la escritura formal de dos algoritmos (sección 3) y un análisis experimental de la complejidad de cada uno de ellos (sección **??**).

2. Formalización del problema

Cuando se piensa en *invertir una cadena* la solución inmediata puede ser muy simplista: inocentemente, se piensa en leer una cadena de atrás hacia adelante. Sin embargo, con un poco más de reflexión, hay tres preguntas que pueden surgir:

1. ¿Cadena es únicamente de tipo `String`?
2. ¿Cómo se guardan esas cadenas en memoria?
3. ¿Solo se pueden ordenar letras?

Recordemos que, en memoria, una cadena de tipo `String` es un arreglo de datos de tipo caracter. Además, es necesario entender que cualquier caracter utilizado por una máquina en realidad tiene un valor `ASCII`. Esto significa que cualquier letra, símbolo o número representado por una máquina, se puede escribir como una secuencia de caracteres. Basado en esta premisa, entendemos que convirtiendo cualquier dato en un arreglo de caracteres, podemos invertir su secuencia y obtener un nuevo dato.

2.1. Definición del problema de la "inversión de la cadena"

Así, el problema de invertir se define a partir de:

1. una secuencia S de elementos $a \in \mathbb{N}$

producir una nueva secuencia S' cuyos elementos cumplan con la relación $a_n,\ a_{n-1},\ ...\ a_0$.

- Entradas:
  - $S = \langle a_i \in \mathbb{N} \rangle \mid 1 \leq i \leq n$.
- Salidas:
  - $S' = \langle e_i \in Sm \rangle \mid e_n,\ e_{n-1} \forall i \in [1, n)$.

3. Algoritmos de solución

3.1. Iterativo

La idea de este algoritmo es: invertir la cadena recorriendo todos elementos desde la última hasta la primera posición.

Algoritmo 1 Binario inverso Iterativo: Convertir un entero a cadena de caracteres binarios.

Require: $n \in \mathbb{N}$
Ensure: $S = \langle e_i \in S \mid e_i < e_{i+1} \forall i \in [1, n) \rangle$

```
1: procedure DecimalToBinary(n)
2:     binario ← []
3:     while n! = 0 do
4:         binario ← (n %2)
5:         n// = 2
6:     end while
7:     return binario
8: end procedure
```

Algoritmo 2 Binario inverso Iterativo: Convertir una cadena de caracteres binarios a un entero.

Require: $S = \langle e_i \in S \mid e_i < e_{i+1} \forall i \in [1, n) \rangle$
Ensure: $n \in \mathbb{N}$

```
1: procedure BinaryToDecimal(binario)
2:     decimal ← 0
3:     for pos, digitoString to enumerate(binario[:: −1]) do
4:         decimal ← Cast(digitoString) ∗ 2 ∗ ∗pos
5:     end for
6:     return decimal
7: end procedure
```

Algoritmo 3 Binario inverso Iterativo

Require: $S = \langle a_i \in \mathbb{N} \rangle \mid 1 \leq i \leq n$
Ensure: $S' = \langle e_i \in Sm \rangle \mid e_n,\ e_{n-1} \forall i \in [1, n)$

```
1: procedure InvertChain(S)
2:     if 1 = |S| then
3:         return S_1
4:     end if
5:     result ← []
6:     for i ← 1 to |S| do
7:         result ← (S[|S| − i])
8:     end for
9:     return result
10: end procedure
```

3.1.1. Análisis de complejidad

Por inspección de código: hay un ciclos *para-todo* que, recorre toda la secuencia de datos para invertirla; entonces, este algoritmo es $O(|S|)$.

3.1.2. Invariante

Después de cada iteración controlada, el elemento i queda en el lugar invertido que le corresponde, es decir $|S| - 1$.

1. Inicio: $i = 0$, la secuencia vacía está invertida.
2. Iteración: $1 \leq i < |S|$, si se supone que los $i - 1$ elementos ya están en su posición, entonces la nueva iteración llevará los i-ésimo elementos a su posición adecuada, es decir al inicio de la secuencia.

3. Terminación: $i = |S|$, los $|S|$ elementos están en su posición, entonces la secuencia está invertida completamente.

3.2. Dividir y vencer

La idea de este algoritmo es: en primera medida, entender que la inversión de una cadena con un único elemento, es igual a la cadena inicial, este siendo el caso base. Por otra parte, eligiendo un pivote en el medio de la secuencia, permite dividirla en subsecuencias de carácteres, que a su vez, son invertidas de derecha a izquierda.

Algoritmo 4 Binario inverso DyV

Require: $S = \langle a_i \in \mathbb{N} \rangle \mid 1 \leq i \leq n$
Ensure: $S' = \langle e_i \in Sm \rangle \mid e_n,\ e_{n-1} \forall i \in [1, n)$

```
1:  procedure InvertChain(S, l, r)
2:      if l = r then
3:          return S_l
4:      end if
5:      q ← (l + r)//2
6:      left ← InvertChain(S,l,q)
7:      right ← InvertChain(S,q+1,r)
8:      for i ← 1 to |right| do
9:          result ← (right[i])
10:     end for
11:     for i ← 1 to |left| do
12:         result ← (left[i])
13:     end for
14:     return result
15: end procedure
```

3.2.1. Análisis de complejidad

El algoritmo ¨InvertChain de dividir y vencer¨ tiene orden de complejidad $O(1)$ cuando hay un elemento en la secuencia S, es decir $n = 1$.

Cuando $n > 1$ la complejidad del algoritmo según el teorema maestro está dada por:

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

El valor de $a = 2$
El valor de $b = 2$
La complejidad de $C(n) = O(1)$

Nos interesa conocer $log_b(a)$, que es $log_2(2) = 1$

Por lo que $f(n) = O(1)$ se compara contra n. En esta situación se aplica el primer caso del teorema maestro donde: $f(n) \in O(n^{log_b^{(a-\epsilon)}}) \longrightarrow T(n) \in \theta(n^{log_b^a})$, asi tambien se define que $f(n)$ esta siendo acotada superiormente por $O(n)$.

Por ende, el orden de complejidad de este algoritmo tambien es:

$$T(n) \in \theta(n)$$

3.2.2. Invariante

La invariante esta dada por: en cada paso de división se realiza el intercambio de la secuencia desde el pivote hacia los extremos.