Tarea 1 - Análisis de algoritmos

David Rivera Morales

February 16, 2025

1 Ejercicio 1

Dado un entero positivo n, determinar el valor de $\lfloor \log(n) \rfloor$.

Solución

Algoritmo

Algorithm 1 Calcular $\lfloor \log(n) \rfloor$

Require: Un entero positivo n
Ensure: El valor de $\lfloor \log(n) \rfloor$

$count \leftarrow 0$
if $n \leq 1$ **then**
  return 0
end if
while $n > 1$ **do**
  $n \leftarrow n/2$
  $count \leftarrow count + 1$
end while
return $count$

Análisis de complejidad

Complejidad Temporal: El algoritmo realiza divisiones sucesivas entre 2 hasta llegar a 1:

- En cada iteración del ciclo **while**, la variable n se divide entre 2
- El número de iteraciones es aproximadamente $\log_2(n)$
- Cada iteración tiene un costo constante $O(1)$

Por lo tanto, la complejidad temporal total es:

$$O(\log n)$$

Complejidad Espacial: El algoritmo utiliza un número constante de variables adicionales ($count$ y n). Por ello, la complejidad espacial es:

$$O(1)$$

En resumen, la complejidad temporal del algoritmo es $O(\log n)$ y la complejidad espacial es $O(1)$.

2 Ejercicio 2

Dado un arreglo A de n enteros y un entero objetivo K, ¿existen un par de índices $i \neq j$, tales que $A[i] + A[j] = K$?

Solución

Algoritmo

Algorithm 2 Encontrar par de números que suman K

Require: Un arreglo A de n enteros y un entero objetivo K
Ensure: Verdadero si existe un par de índices $i \neq j$ tales que $A[i] + A[j] = K$

```
hashMap ← {}
for i ← 0 to n − 1 do
    if hashMap.containsKey(K − A[i]) then
        return true
    end if
    hashMap.put(A[i], i)
end for
return false
```

Análisis de complejidad

Complejidad Temporal: El algoritmo realiza las siguientes operaciones:

- Recorre el arreglo una sola vez, visitando sus n elementos
- Para cada elemento realiza operaciones de hash (búsqueda e inserción) que son $O(1)$ en promedio

Por lo tanto, la complejidad temporal total es:

$$O(n)$$

Complejidad Espacial: El algoritmo utiliza un `hashMap` que puede almacenar hasta n elementos. Por ello, la complejidad espacial es:

$$O(n)$$

Resumen: - **Tiempo Promedio:** $O(n)$ - **Tiempo Peor Caso:** $O(n^2)$ (escenario teórico con colisiones excesivas) - **Espacio:** $O(n)$

En conclusión, el algoritmo es eficiente en la mayoría de los casos prácticos, logrando encontrar el par de números que suman K en tiempo lineal con respecto al tamaño del arreglo.

3 Ejercicio 3

Dado un entero positivo n, determinar la cantidad de números primos menores o iguales a n.

Solución

Algoritmo

Algorithm 3 Contar números primos usando la Criba de Eratóstenes

Require: Un entero positivo n
Ensure: La cantidad de números primos menores o iguales a n

$esPrimo \leftarrow [true] * (n+1)$ {Arreglo booleano inicializado en verdadero}
$esPrimo[0] \leftarrow false$
$esPrimo[1] \leftarrow false$
$contador \leftarrow 0$
for $i \leftarrow 2$ **to** $\sqrt{n}$ **do**
  if $esPrimo[i]$ **then**
    for $j \leftarrow i^2$ **to** n **step** i **do**
      $esPrimo[j] \leftarrow false$
    end for
  end if
end for
for $i \leftarrow 2$ **to** n **do**
  if $esPrimo[i]$ **then**
    $contador \leftarrow contador + 1$
  end if
end for
return $contador$

Análisis de complejidad

Complejidad Temporal: El algoritmo realiza las siguientes operaciones:

- Inicialización del arreglo: $O(n)$
- Marcado de múltiplos: $O(n \log \log n)$
- Conteo final de primos: $O(n)$

Por lo tanto, la complejidad temporal total es:

$$O(n \log \log n)$$

Complejidad Espacial: El algoritmo utiliza un arreglo booleano de tamaño $n + 1$. Por ello, la complejidad espacial es:

$$O(n)$$

4 Ejercicio 5

Dado un arreglo A de n enteros, ¿existe un elemento de A tal que aparece en A al menos $n/2$ veces?

Solución

Algoritmo

Algorithm 4 Encontrar elemento mayoritario (apariciones $\geq n/2$)

Require: Un arreglo A de n enteros.
Ensure: Un elemento que aparece al menos $n/2$ veces, o null si no existe.

```
1: candidate ← null
2: count ← 0
3: for cada x en A do
4:    if count = 0 then
5:       candidate ← x
6:       count ← 1
7:    else
8:       if candidate = x then
9:          count ← count + 1
10:      else
11:         count ← count − 1
12:      end if
13:   end if
14: end for
15: occurrence ← 0
16: for cada x en A do
17:   if x = candidate then
18:      occurrence ← occurrence + 1
19:   end if
20: end for
21: if occurrence ≥ n/2 then
22:   return candidate
23: else
24:   return null {No existe elemento mayoritario}
25: end if
```

Análisis de complejidad

Complejidad Temporal: El algoritmo realiza dos recorridos sobre el arreglo A:

- El primer recorrido (líneas 3 a 13) para identificar un candidato mayoritario, con una complejidad de $O(n)$.
- El segundo recorrido (líneas 14 a 18) para verificar que el candidato realmente aparece al menos $n/2$ veces, también con una complejidad de $O(n)$.

Por lo tanto, la complejidad temporal total es:

$$O(n) + O(n) = O(n)$$

Complejidad Espacial: El algoritmo utiliza únicamente un número constante de variables adicionales (*candidate*, *count* y *occurrence*). Por ello, la complejidad espacial es:

$$O(1)$$