Teoría de Algoritmos

Curso Echevarría

Trabajo Práctico 0

2° Cuatrimestre 2025

Alumna: Damaris Juares
Padrón: 108566

Índice

1. Supuestos **3**
1.1. Primicias 3
1.2. Limitaciones 3
1.3. Condiciones 3

2. Diseño **3**
2.1. Estructura 3
2.2. Solución 3

3. Análisis de complejidad: **4**

4. Seguimiento **4**

5. Tiempos de ejecución **5**

6. Informe de resultados **6**

7. Alternativas **7**

1. Supuestos

1.1. Primicias

La primicia para buscar divisores de un numero es:

Un múltiplo propio de un número x, es un número mayor que x y divisible por x.

Siguiendo esto, el algortimo inicialmente para un valor n deberá buscar los múltiplos de todos los valores i, con i= 0,1,2,.., n. Y almacenar i para cada valor encontrado como su divisor.

1.2. Limitaciones

Almacenar la suma de los divisores en una estructura de datos consume $\mathcal{O}(n)$ de la memoria.

1.3. Condiciones

Un numero i es amigo de otro numero s si:

s = sumasDivisores[i] y
i= sumasDivisores[s]

Y adicionalmente para evitar pares de amigos repetidos se agrega la siguiente restricción:

Sean numero1 y numero2 los dos valores a analizar: numero2 $\leq numero1$

2. Diseño

2.1. Estructura

Se utiliza una lista de enteros: sumasDivisores

sumasDivisores = [0] * (MAX + 1)

Donde MAX es el valor de entrada en la funcion.

2.2. Solución

A continuación se muestra la solución propuesta.

```
import time
import math

def amigos_optimizado(MAX):
    t1 = time.time()

    # Registrar sumas de divisores
    sumas_divisores = [0] * (MAX + 1)

    # Buscar divisores a partir de los multiplos
    for numero in range(1, MAX + 1):
        for multiplo in range(numero * 2, MAX + 1, numero):
            sumas_divisores[multiplo] += numero

    # Buscar pares de numeros amigos
    for numero1 in range(MAX+1) :
        numero2 = sumas_divisores[numero1]
```

```
18
19          # Verificar condiciones para n meros amigos
20          if numero2 <= numero1 and sumas_divisores[numero2] == numero1:
21              print(numero1,numero2)
22
23      t2 = time.time()
24      print(t2-t1)
```

3. Análisis de complejidad:

El bucle interno se ejecuta n/i veces para $i = 1, 2, 3, 4,$, lo que hace que el número total de operaciones en el bucle interno sea una suma armónica como $n(1+1/2+1/3+1/4+\cdots)$, que está delimitado por $\mathcal{O}(nlogn)$

4. Seguimiento

Se detalla el seguimiento con los números del 215 al 225.

El algoritmo actualiza la estructura de datos inicializada en 0 que contiene las sumas de los divisores de todos los números en el rango de la entrada n (incluyendo n).

Las sumas de divisores propios para cada número analizado son:

i: 215 - suma: 49

i: 216 - suma: 384

i: 217 - suma: 39

i: 218 - suma: 112

i: 219 - suma: 77

i: 220 - suma: 284

i: 221 - suma: 31

i: 222 - suma: 234

i: 223 - suma: 1

i: 224 - suma: 280

i: 225 - suma: 178

Verificación para cada i si es parte de un par de números amigos, con s= suma:

Para i=215:

s = 49

¿s $\leq$ 225?

sumasDivisores[49] = 8

¿8 == 215? No $\rightarrow$ No son amigos

Para i=216:

s = 384

¿s $\leq$ 216?

Para i=217:

s = 39

¿s $\leq$ 217?

sumasDivisores[39] = 17

¿17 == 217? No $\rightarrow$ No son amigos

Para i=218:

s = 112

¿s $\leq$ 218?

sumasDivisores[112] = 136

¿136 == 218? No $\rightarrow$ No son amigos

Para i=219:

s = 77

¿s $\leq$ 219?

sumasDivisores[77] = 19

¿19 == 219? No $\rightarrow$ No son amigos

Para i=220:

s = 284

¿s $\leq$ 220?$No ß Descartar$

Para i=221:

s = 31

¿s $\leq$ 225?

sumasDivisores[31] = 1

¿1 == 221? No $\rightarrow$ No son amigos

Para i=222:

s = 234

¿s $\leq$ 225?$No ß Descartar$

Para i=223:

s = 1

¿s $\leq$ 225?

sumasDivisores[1] = 0

¿0 == 223? No $\rightarrow$ No son amigos

Para i=224:

s = 280

¿s $\leq$ 225?

Para i=225:

s = 178

¿s $\leq$ 225?

sumasDivisores[178] = 92

¿92 == 225? No $\rightarrow$ No son amigos

5. Tiempos de ejecución

A continuación se presenta la medición y gráfica de los tiempos de ejecución para los números de 1 a 50000, 1 a 100000, 1 a 150000, 1 a 200000 y 1 a 250000.

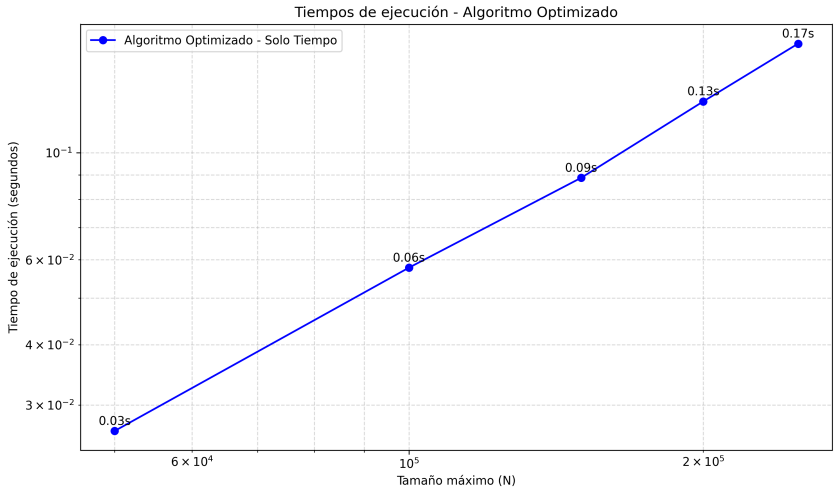

Figura 1: Mediciones temporales

6. Informe de resultados

Informe de Resultados - Algoritmo Optimizado

1. **Resumen del análisis**

   El gráfico muestra los *tiempos de ejecución* del algoritmo optimizado para la búsqueda de números amigos en función del tamaño máximo N. Los puntos representados son:

Tamaño máximo (N)	Tiempo de ejecución (s)
50,000	0.03
100,000	0.06
150,000	0.09
200,000	0.13
250,000	0.17

   Se observa un crecimiento casi lineal, como indica la progresión casi uniforme en los tiempos al duplicar o aumentar proporcionalmente el valor de N.

2. **Comparación con la complejidad estimada**

   Se había estimado que el algoritmo tiene una *complejidad temporal* de $O(n \log n)$, lo cual es consistente con el comportamiento observado en el gráfico:

   - En el eje Y (tiempo), se utiliza una escala logarítmica, lo cual permite identificar tendencias exponenciales o logarítmicas más fácilmente.
   - La forma casi lineal en el gráfico (cuando el eje Y es logarítmico) sugiere que el crecimiento no es cuadrático ($O(n^2)$) ni lineal puro ($O(n)$), sino sublineal respecto a un cuadrado, como $O(n \log n)$.

3. **Conclusiones**

   - El algoritmo optimizado muestra una escalabilidad eficiente, manteniendo tiempos bajos incluso para valores grandes de N.
   - El crecimiento de los tiempos de ejecución está en línea con una complejidad $O(n \log n)$, lo que indica que la optimización ha sido efectiva.

7. Alternativas

Alternativa 1: Uso de propiedades matemáticas (generación directa de pares)

Una fórmula atribuida a *Thābit ibn Qurra* genera pares de números amigos (no todos, pero algunos). La fórmula es:

$$\text{Si} \quad p = 3 \cdot 2^n - 1, \quad q = 3 \cdot 2^{n-1} - 1, \quad r = 9 \cdot 2^{2n-1} - 1,$$

y si p, q y r son números primos, entonces los números

$$2^n \cdot p \cdot q \quad \text{y} \quad 2^n \cdot r$$

forman un par de números amigos.