Tarea 1: Análisis y Diseño de Algoritmos

Pontificia Universidad Javeriana Cali
Juan Diego Collazos

9 de febrero de 2025

Código de Honor

Como miembro de la comunidad académica de la Pontificia Universidad Javeriana Cali me comprometo a seguir los más altos estándares de integridad académica. Integridad académica se refiere a ser honesto, dar crédito a quien lo merece y respetar el trabajo de los demás. Por eso es importante evitar plagiar, engañar, 'hacer trampa', etc. En particular, el acto de entregar un programa de computador ajeno como propio constituye un acto de plagio; cambiar el nombre de las variables, agregar o eliminar comentarios y reorganizar comandos no cambia el hecho de que se está copiando el programa de alguien más. Para más detalles consultar el Reglamento de Estudiantes, Sección VI.

Growth rate

Considera las siguientes funciones y organizarlas en orden ascendente de tasa de crecimiento. Si $g(n)$ sigue inmediatamente a $f(n)$ en la lista, entonces debe ser el caso que $f(n) = O(g(n))$.

- $f_1(n) = n^{2,5}$
- $f_2(n) = \sqrt{2n}$
- $f_3(n) = n + 10$
- $f_4(n) = 10^n$
- $f_5(n) = 100^n$
- $f_6(n) = n^2 \log n$

Solución

Ordenemos las funciones en orden ascendente de tasa de crecimiento:

$$f_2(n) = \sqrt{2n} < f_3(n) = n+10 < f_6(n) = n^2 \log n < f_1(n) = n^{2,5} < f_4(n) = 10^n < f_5(n) = 100^n.$$

Demostraciones

Definición de Big O

$$O(f) = \{g : \mathbb{N} \to \mathbb{R}^+ \mid \exists n_0 \in \mathbb{N}, \exists c \in \mathbb{R}^+, \forall n \in \mathbb{N}, n \geq n_0 \implies g(n) \leq c \cdot f(n)\}.$$

Teorema 1: $\sqrt{2n} \in O(n + 10)$

Demostración: Se procede de manera directa a partir de la definición de Big O. De acuerdo con esta definición, es necesario encontrar constantes c y n_0 tales que:

$$\forall n \geq n_0 \implies f(n) \leq c \cdot g(n).$$

$$\sqrt{2n} \leq c \cdot n + 10.$$

$$\frac{\sqrt{2n}}{n + 10} \leq c$$

El lado izquierdo de la desigualdad se analiza mediante límites:

$$\lim_{x \to \infty} \frac{\sqrt{2x}}{x + 10}$$

Aplicando la regla de L'Hôpital debido a que el cociente tiene la forma indeterminada $\frac{\infty}{\infty}$:

$$\lim_{x \to \infty} \frac{\sqrt{2x}}{x + 10} = \lim_{x \to \infty} \frac{\frac{d}{dx}(\sqrt{2x})}{\frac{d}{dx}(x + 10)}. = \lim_{x \to \infty} \frac{\frac{2}{\sqrt{2x}}}{1} = \lim_{x \to \infty} \frac{2}{\sqrt{2x}} = 0.$$

Como el límite del cociente es 0, esto implica que, para valores suficientemente grandes de n, existe una constante $c > 0$ tal que $\sqrt{2n} \leq c \cdot (n + 10)$. Podemos tomar cualquier $c > 0$ y encontrar un n_0 tal que la desigualdad se cumpla para todo $n \geq n_0$. Por esto se concluye que $\sqrt{2n} \leq O(n + 10)$ tomando $c = 1$ y $n_0 = 1$ como testigos.

Teorema 2: $n + 10 \in O(n^2 \log n)$

Demostración: Se procede de manera directa a partir de la definición de Big O. De acuerdo con esta definición, es necesario encontrar constantes c y n_0 tales que:

$$\forall n \geq n_0 \implies f(n) \leq c \cdot g(n).$$

$$n + 10 \leq c \cdot n^2 \log(n)$$

$$\frac{n + 10}{n^2 log(n)} \leq c$$

$$\frac{n}{n^2 log(n)} + \frac{10}{n^2 log(n)} \leq c$$

$$\frac{1}{n log(n)} + \frac{10}{n^2 log(n)} \leq c$$

Es posible notar que el valor más grande que puede tomar $\frac{1}{n\log(n)}+\frac{10}{n^2\log(n)}$ es 11 cuando $n\log(n)$ y $n^2\log(n)$ se aproximan a 1. Por lo tanto, se concluye que $n+10 \in O(n^2 \log n)$ con testigos $c = 11$ y $n_0 = 2$.

Teorema 3: $n^2 \log n \in O(n^{2,5})$

Demostración: Se procede de manera directa a partir de la definición de Big O. De acuerdo con esta definición, es necesario encontrar constantes c y n_0 tales que:

$$\forall n \geq n_0 \implies f(n) \leq c \cdot g(n).$$

$$n^2 \log(n) \leq c \cdot n^{2,5}$$

$$\frac{n^2 \log(n)}{n^{2,5}} \leq c$$

$$\frac{n^2 \log(n)}{n^2\sqrt{n}} \leq c$$

$$\frac{\log(n)}{\sqrt{n}} \leq c$$

El lado izquierdo de la desigualdad se analiza mediante límites:

$$\lim_{x\to\infty} \frac{\log(x)}{\sqrt{x}}$$

Aplicando la regla de L'Hôpital debido a que el cociente tiene la forma indeterminada $\frac{\infty}{\infty}$:

$$\lim_{x\to\infty} \frac{\log(x)}{\sqrt{x}} = \lim_{x\to\infty} \frac{\frac{d}{dx}(\log(x))}{\frac{d}{dx}(\sqrt{x})} = \lim_{x\to\infty} \frac{2\sqrt{x}}{x} = \lim_{x\to\infty} \frac{2}{\sqrt{x}} = 0$$

Como el límite del cociente es 0, esto implica que, para valores suficientemente grandes de n, existe una constante $c > 0$ tal que $n^2 \log(n) \leq c \cdot n^{2,5}$. Podemos tomar cualquier $c > 0$ y encontrar un n_0 tal que la desigualdad se cumpla para todo $n \geq n_0$. Por esto se concluye que $n^2 \log n \in O(n^{2,5})$ tomando $c = 1$ y $n_0 = 2$ como testigos.

Teorema 4: $n^{2,5} \in O(10^n)$

Demostración: Se procede de manera directa a partir de la definición de Big O. De acuerdo con esta definición, es necesario encontrar constantes c y n_0 tales que:

$$\forall n \geq n_0 \implies f(n) \leq c \cdot g(n).$$

$$n^{2,5} \leq c \cdot 10^n$$

$$\frac{n^{2,5}}{10^n} \leq c$$

El lado izquierdo de la desigualdad se analiza mediante límites:

$$\lim_{x\to\infty} \frac{x^{2,5}}{10^x}$$

La regla de L'Hôpital puede aplicarse repetidamente para el caso de potencias y exponenciales, y al final obtenemos que:

$$\lim_{x\to\infty} \frac{x^{2,5}}{10^x} = 0$$

Como el límite del cociente es 0, esto implica que, para valores suficientemente grandes de n, existe una constante $c > 0$ tal que $n^{2,5} \leq c \cdot 10^n$. Podemos tomar cualquier $c > 0$ y encontrar un n_0 tal que la desigualdad se cumpla para todo $n \geq n_0$. Por esto se concluye que $n^{2,5} \in O(10^n)$ tomando $c = 1$ y $n_0 = 2$ como testigos.

Teorema 5: $10^n \in O(100^n)$

Demostración: Se procede de manera directa a partir de la definición de Big O. De acuerdo con esta definición, es necesario encontrar constantes c y n_0 tales que:

$$\forall n \geq n_0 \implies f(n) \leq c \cdot g(n).$$

$$10^n \leq c \cdot 100^n$$

$$\frac{10^n}{100^n} \leq c$$

$$\left(\frac{10}{100}\right)^n \leq c$$

$$\left(\frac{1}{10}\right)^n \leq c$$

$$\frac{1}{10^n} \leq c$$

Es posible notar que el valor más grande que puede tomar $\frac{1}{10^n}$ es 1 cuando $n = 0$. Por lo tanto, se concluye que $10^n \in O(100^n)$ con testigos $c = 1$ y $n_0 = 1$.

Text segmentation problem

Describa algoritmos recursivos para las siguientes variantes del problema de segmentación de texto. Suponga que tiene una subrutina IsWord que toma una arreglo de caracteres como entrada y devuelve True si y solo si esa cadena es una "palabra".

The number of partitions of A into words

1. Especificación del problema

Entrada: Un arreglo $A[0..n)$ de caracteres, donde se garantiza que cada carácter de A pertenece a una subsecuencia que sea una palabra válida según IsWord.

Salida: Un número natural que representa la cantidad de formas diferentes de dividir A en subsecuencias consecutivas de caracteres, donde cada subsecuencia es válida según IsWord.

2. Función objetivo

Sea $P(n)$ el número de particiones del arreglo de caracteres A[0..n) en subsecuencias que sean palabras válidas.

3. Reformulación de la salida

Para cada índice i en 0 a $n-1$, si A[0..i] es una palabra válida (IsWord(A[0..i])), entonces el número total de particiones será la suma de las particiones válidas del resto de la cadena, A[i+1..n).

4. Planteamiento recurrente

La recurrencia se define como:

$$P(n) = \begin{cases} 1 & \text{si } n = 0 \\ \sum_{i=0}^{n-1} \text{IsWord}(A[0..i]) \cdot P(n-(i+1)) & \text{si } n > 0 \end{cases}$$

Aquí:

- IsWord$(A[0..i])$ devuelve 1 si $A[0..i]$ es una palabra válida y 0 en caso contrario.
- El caso base $P(0) = 1$ representa la única partición válida para la cadena vacía.

5. Análisis de complejidad (Teorema Maestro)

La relación de recurrencia para el algoritmo es:

$$T(n) = \sum_{i=0}^{n-1} T(n-(i+1)) + O(n)$$

Equivalente a $T(n) = 2^n$ (al ser una recurrencia exponencial). Esto da una complejidad de:

$$O(2^n)$$

En nuestro caso, la relación de recurrencia no se ajusta directamente al formato del teorema maestro porque el número de subproblemas no es constante (aquí tenemos una suma acumulativa). Sin embargo, podemos concluir que $T(n)$ crece más rápido que cualquier relación de tipo $T(n) = aT(n/b) + O(n^d) T(n) = aT(n/b) + O(n^d)$, indicando que el crecimiento es exponencial.

6. Algoritmo

```
1 def CountPartitions(A, n):
2     ans = 1
3     if n != 0:
4         ans = 0
```

```
        for i in range(1, n + 1):
            if IsWord(A[:i]):  # Subcadena A[0..i-1]
                ans += CountPartitions(A[i:], n - i)
    return ans
```

Decide whether A and B can be partitioned into words at the same indices

1. Especificación del problema

Entrada: Dos arreglos de caracteres $A[0..n)$ y $B[0..n)$, donde se garantiza que cada carácter de A o B pertenece a una subsecuencia que sea una palabra válida según IsWord.

Salida: Un booleano que representa si existen divisiones en palabras válidas para ambos arreglos usando los mismos índices.

2. Función objetivo

Sea $Q(n)$ una función booleana que devuelve True si $A[0..n)$ y $B[0..n)$ pueden dividirse con los mismos índices en palabras válidas; de lo contrario, devuelve False.

3. Reformulación de la salida

Para cada índice i en 0 a $n-1$, si ambas subcadenas $A[0..i]$ y $B[0..i]$ son palabras válidas, verificamos recursivamente si las subsecuencias restantes, $A[i+1..n)$ y $B[i+1..n)$, también pueden dividirse con los mismos índices.

4. Planteamiento recurrente

La recurrencia se define como:

$$Q(n) = \begin{cases} \text{True} & \text{si } n = 0 \\ \bigvee_{i=0}^{n-1} \left(\text{IsWord}(A[0..i]) \wedge \text{IsWord}(B[0..i]) \wedge Q(n-(i+1))\right) & \text{si } n > 0 \end{cases}$$

Donde el caso base $Q(0) = \text{True}$ indica que las cadenas vacías pueden dividirse con los mismos índices.

5. Análisis de complejidad (Teorema Maestro)

La relación de recurrencia para el algoritmo es similar al caso (a):

$$T(n) = \sum_{i=0}^{n-1} T(n-(i+1)) + O(n)$$

Esto también conduce a una complejidad exponencial:

$$O(2^n)$$

6. Algoritmo

```python
def CanSplitSameIndices(A, B, n):
    if n == 0:
        return True
    for i in range(1, n + 1):
        if IsWord(A[:i]) and IsWord(B[:i]):  # Subcadenas A[0..i-1] y B[0..i-1]
            if CanSplitSameIndices(A[i:], B[i:], n - i):
                return True
    return False
```

Cantidad de formas diferentes en que A y B se pueden dividir en palabras con los mismos índices

1. Especificación del problema

Entrada: Dos arreglos de caracteres $A[0..n)$ y $B[0..n)$, donde se garantiza que cada carácter de A o B pertenece a una subsecuencia que sea una palabra válida según IsWord.

Salida: Un numero natural que representa la cantidad de formas en que ambos pueden dividirse simultáneamente en palabras válidas utilizando los mismos índices.

2. Función objetivo

Sea $R(n)$ el número de formas diferentes en que $A[0..n)$ y $B[0..n)$ pueden dividirse en palabras válidas con los mismos índices.

3. Reformulación de la salida

Para cada índice i en 0 a $n-1$, si $A[0..i]$ y $B[0..i]$ son palabras válidas, las divisiones totales serán la suma de las divisiones válidas de las subsecuencias restantes $A[i+1..n)$ y $B[i+1..n)$.

4. Planteamiento recurrente

La recurrencia se define como:

$$R(n) = \begin{cases} 1 & \text{si } n = 0 \\ \sum_{i=0}^{n-1} \text{IsWord}(A[0..i]) \cdot \text{IsWord}(B[0..i]) \cdot R(n-(i+1)) & \text{si } n > 0 \end{cases}$$

El caso base $R(0) = 1$ representa la única forma válida para cadenas vacías.

5. Análisis de complejidad (Teorema Maestro)

La relación de recurrencia es nuevamente:

$$T(n) = \sum_{i=0}^{n-1} T(n-(i+1)) + O(n)$$

Esto resulta en una complejidad de:

$$O(2^n)$$

6. Algoritmo

```python
def CountMatchingSplits(A, B, n):
    if n == 0:
        return 1
    ans = 0
    for i in range(1, n + 1):
        if IsWord(A[:i]) and IsWord(B[:i]):  # Subcadenas A[0..i-1] y B[0..i-1]
            ans += CountMatchingSplits(A[i:], B[i:], n - i)
    return ans
```