Solución Parcial 1 - Análisis Algoritmos

Nicolás Camacho-Plazas

25 de septiembre de 2020

Parte I
Algoritmos de promedio

1. Diseño del problema

1.1. Entradas

- Una secuencia $A = \langle a_1, a_2, \cdots, a_n \rangle = \langle a_i \in \mathbb{T} \rangle$, si $n = |A|$, donde el conjunto $\mathbb{T}$ está conformados por los elementos que tienen definidas las operaciones de suma y resta.
- Dos indices i y j, donde $i \wedge J \in [1, ..., |A|]$, que representan los límites de la subsección de la secuencia de donde se pretende encontrar el promedio.

1.2. Salidas

- Un valor $z = \frac{1}{|A|} \sum_1^{|A|} a_i$ que por ende contiene el promedio de los valores de la subsección delimitada entre i y j.

2. Complejidad

2.1. Algoritmo 1 - Promedio inocente

Al realizar inspección de código, por el único for, se concluye que el algoritmo tiene un orden de complejidad de $O(|A|)$.

2.2. Algoritmo 2 - Promedio dividir y vencer

De acuerdo al teorema maestro, y teniendo en cuenta que el problema divide en dos los datos, usa dos veces la recursión y la complejidad fuera de la recursión es $O(1)$, se puede determinar que:

$$T(n) = \begin{cases} O(1), & \text{caso base } n <= 1, \\ 2T(\frac{n}{2}) + O(1), & \text{si } n \in |A| \end{cases}$$

De modo que al estudiar el caso uno del teorema maestro, se establece que: $O(1) = O(n^{\log_2 2-\epsilon})$

$\therefore \log_2 2 - \epsilon = 0 \wedge \epsilon = 1$

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

3. Invariante

3.1. Promedio inocente

μ siempre va a ser el total de la secuencia hasta $A[k]$ donde $k \in [1, ..., |A|]$.

3.2. Dividir y conquistar

- x siempre contiene el promedio parcial de la secuencia contenida entre i y $q - 1$.
- y siempre contiene el promedio parcial de la secuencia contenida entre $q + 1$ y j.

3.3. De acuerdo a los puntos anteriores, ?cuál algoritmo es mejor de implementar en un contexto dónde se deba calcular muchos promedios en conjuntos de datos de tamaños del orden de los billones de elementos?

Teniendo en cuenta que una operación de división es más compleja de realizar que una suma o resta y el algoritmo inocente solo realiza una por promedio, la pila de ejecuciones que genera una recursión tan extensa, y la cantidad de variables empleadas en la recursión, se concluye que resulta más conveniente utilizar el algoritmo de "Promedio inocente".

Parte II
Escritura de un algoritmo

4. Análisis del problema

Dada una secuencia A de elementos A_i donde $A_i \in \mathbb{T}$ e $i \in [1, ..., |A|]$, cuyo conjunto $\mathbb{T}$ está conformado por elementos con los que es posible utilizar la operación "$<$", y unos límites i y j que cumplen $i \wedge j \in [1, ..., |A|] \wedge i < j$, se busca encontrar la cantidad y límites de los rangos aditivos presentes en dicha secuencia. Un rango aditivo se define como una pareja de índices $[a, b]$ donde $A[a] > A[i] \wedge A[a] < A[j]$ y $A[b] > A[i] \wedge A[b] < A[j]$.

5. Diseño del Algoritmo

5.1. Entradas

- Una secuencia A descrita en el análisis del problema y de donde se quieren.
- Un indice i que representa el valor menor de la pareja para encontrar los rangos aditivos y donde $i \in [1, ..., j-1]$.
- Un indice $j > i$ que representa el valor mayor de la pareja para encontrar los rangos aditivos y donde $j \in [n, ..., |A|]$ donde $n > i$.

5.2. Salidas

- Un valor z que representa la cantidad de rangos aditivos que cumplen con la pareja i, j dentro de la secuencia A.

6. Algoritmo

Para resolver el algoritmo se utilizan dos algoritmos, «CountAditiveRangesAux» en el que se realiza la recursión siguiendo el paradigma de dividir y vencer, y «CountAditiveRanges» el cual adapta los parámetros del usuario y adecúa el retorno.

Algorithm 1 Número posibles límites de rangos aditivos - Dividir y vencer

```
1:  procedure CountAditiveRangesAux(A,b, e, range)
2:      if b = e then
3:          if range[1] < A[b] and A[b] < range[2] then
4:              return 1
5:          else
6:              return 0
7:          end if
8:      else
9:          q = ⌊(e + b)/2⌋
10:         totalLeft = CountAditiveRangesAux(A, b, q, range)
11:         totalRight = CountAditiveRangesAux(A, q + 1, e, range)
12:         return totalLeft + totalRight
13:     end if
14: end procedure
```

El siguiente algoritmo utiliza un método llamado *Combinatorial* que puede ser cualquier método de cualquier librería que calcule la combinatoria de dos números enteros.

Algorithm 2 Número de rangos aditivos en la secuencia

1: **procedure** CountAditiveRanges(A, i, j)
2: $total = CountAditiveRangesAux(A, 1, |A|, [i, j])$
3: **return** $\frac{total!}{2(total-2)!} + total$
4: **end procedure**

7. Análisis de complejidad

Al utilizar el teorema maestro, teniendo en cuenta que se divide en 2 el problema y se realizan dos llamados recursivos, se establece que:

$$T(n) = \begin{cases} O(1), & \text{caso base } n = 1, \\ 2T(\frac{n}{2}) + O(1), & \text{si } n \in |A| \end{cases}$$

De modo que al estudiar el caso uno del teorema maestro, se establece que: $O(1) = O(n^{\log_2 2-\epsilon})$
$\therefore \log_2 2 - \epsilon = 0 \wedge \epsilon = 1$
$\therefore T(n) \in \theta(n)$

8. Invariante

- *CountAditiveRangesAux*: «$totalLeft$» siempre va a tener el total de los elementos que están dentro de el rango comprendido entre i y j de la subsecuencia delimitada por b y q. «*totalRight*» siempre va a tener el total de los elementos que están dentro de el rango comprendido entre i y j de la subsecuencia delimitada por $q + 1$ y e.
- *CountAditiveRanges*: «*total*» contiende la cantidad de elementos que pueden representar los límites de los rangos aditivos. Su combinatoria contempla la cantidad de rangos que se pueden formar entre dichos elementos y al sumar el total se contemplan los casos en los que el rango comience y termine por el mismo elemento.