# Taller 3: Multiplicación matricial

Alejandro Morales Contreras[1]

[1]Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
a.moralesc@javeriana.edu.co

1 de septiembre de 2022

# Índice



# 1. Enunciado del problema

Escribir formalmente, implementar en Julia y hacer experimentos sobre un algoritmo basado en programación dinámica que resuelva el siguiennte problema:

Calcular la parentización que minimice la cantidad de multiplicaciones escalares en una composición de matrices.

# 2. Formalización del problema

Sean $A \in \mathbb{R}^{n\times m}$ y $B \in \mathbb{R}^{m\times p}$ matrices (nótese que la segunda dimensión de $A$ es igual a la primera dimensión de $B$), la multiplicación matricial $AB$ da como resultado una nueva matriz $C \in \mathbb{R}^{n\times p}$. La cantidad de multiplicaciones escalares necesarias para generar $C$ equivale a $n \cdot m \cdot p$.

Sea $\mathbb{A} = \langle A_1, A_2, \cdots, A_n\rangle$ una secuencia de matrices que cumplen la relación $A_i \in \mathbb{R}^{r_i\times c_i} \wedge r_i = c_{i-1}$, es importante saber cuál es la mejor forma de asociar, de dos en dos, las multiplicaciones matriciales para reducir la cantidad de multiplicaciones escalares. Esto debido a que la agrupación, de acuerdo a las dimensiones de las matrices vecinas, puede generar resultados muy diferentes. Es necesario entonces **minimizar** la cantidad de multiplicaciones escalares encontrando la mejor combinación de paréntisis (agrupación) para multiplicar las matrices.

Nótese que $r_i = c_{i-1} \forall i \in [2, n]$, la cantidad de filas de la matriz $A_i$ es igual a la cantidad de columnas de la matriz anterior $A_{i-1}$. Es posible definir $D = \langle d_i \in \mathbb{N} \mid 1 \leq i \leq n+1\rangle$ donde $d_1$ es la cantidad de filas de la primera matriz, $d_2$ es la cantidad de filas de la segunda matriz y la cantidad de columnas de la primera, y en general, $d_i = r_i = c_{i-1}$.

## 2.1. Definición del problema "multiplicación matricial"

El problema de calcular la parentización que minimice la cantidad de multiplicaciones escalares en una composición de matrices se define a partir de $\mathbb{A} = \langle A_1, A_2, \cdots, A_n \mid A_i \in \mathbb{R}^{r_i\times c_i} \wedge r_i = c_{i-1}\rangle$ una secuencia de matrices, $\mathbb{S} = \langle \text{`A'}\rangle \cup \langle \text{`0'}, \text{`1'}, \cdots, \text{`9'}\rangle \cup \langle \text{`('}, \text{`)'}\rangle$ un conjunto de caracteres, encontrar la representación en cadena de caracteres $S$ que ilustre como agrupar los paréntesis para multiplicar dichas matrices minimizando la cantidad de multiplicaciones escalares.

- Entradas:
  - $D = \langle d_i \in \mathbb{N} \mid 1 \leq i \leq n+1 \wedge d_i = r_i = c_{i-1}\rangle$
- Salidas:
  - $S = \langle s_i \in \mathbb{S}\rangle$

# 3. Algoritmo de solución

## 3.1. Idea general de la solución

Asociar las matrices en dos grupos puede ser visto como una tarea de fuerza bruta, donde se define un $k \in [1, n]$ que separa ambos grupos con paréntesis así: $A_{1,k}A_{k+1,n} = (A_1 \cdots A_k)(A_{k+1} \cdots A_n)$, tal que esta asociación minimiza la cantidad de multiplicaciones escalares realizadas.

Debido a que las asociaciones deben llegar hasta un punto que se tengan multiplicaciones de 2 matrices, el paso siguiente es encontrar el siguiente $k$ que minimiza la cantidad de multiplicaciones escalares para el primer grupo, y encontrar el $k$ del segundo grupo. Y en general, la secuencia de matrices se puede asociar desde $A_i$ hasta $A_j$ $(i < j)$ encontrando el $i \leq k < j$ que minimiza la cantidad de multiplicaciones escalares.

Entonces, $M_{i,j}$ es el número óptimo de multiplicaciones escalares al agrupar las matrices $A_{i,j}$ definido así:

$$M_{i,j} = \begin{cases} 0, & \text{if } i = j, \\ \min\limits_{i\leq k<j}\{M_{i,k} + M_{k+1,j} + m_{ikj}\}, & \text{if } i \neq j. \end{cases}$$

donde $m_{ijk}$ es el número de multiplicaciones escalares para calcular $A_{i,k}A_{k+1,j}$.

Dado que: $A_iA_{i+1} \in \mathbb{R}^{r_i\times c_{i+1}}, A_iA_{i+1}A_{i+2} \in \mathbb{R}^{r_i\times c_{i+2}}, \cdots, A_{i,k} \in \mathbb{R}^{r_i\times c_k}, A_{k+1,j} \in \mathbb{R}^{r_{k+1}\times c_j}$ y $r_{k+1} = c_k$, entonces:

$$m_{ikj} = r_i c_k c_j$$

Así mismo, dado que la entrada de la secuencia de matrices $\mathbb{A}$ se puede definir como la secuencia $D$ y por ende $r_i, c_i$ se pueden re-escribir como elementos de $D$, entonces:

$$m_{ikj} = d_{i-1} d_k d_j$$

Finalmente, el problema se puede resolver por fuerza bruta y recursión. Después, se puede optimizar con una solución con programación dinámica por memoización que reduzca el cálculo de operaciones repetidas. Seguidamente, se puede generar una solución bottom-up a partir del algoritmo con memoización para finalmente implementar una solución bottom-up con backtracking para encontrar cuáles son los $k$ donde se ubican los paréntesis para minimizar la cantidad multiplicaciones escalares.

### 3.2. Escritura del algoritmo

Donde:

- $\textsc{Concat}(p_1, p_2, \cdots, p_n)$ recibe $n$ cadenas de caracteres y las une en una sola en orden
- $\textsc{String}(x)$ recibe un valor numérico y retorna una cadena de caracteres como representación

**Algoritmo 1** Calcular y representar parentización que minimiza multiplicaciones escalares.

```
1: procedure MinMatMul(D)
2:     B ← MinMatMul_Backtracking(D)
3:     return DrawParentheses(B, 1, |D| − 1)
4: end procedure
```

**Algoritmo 2** Calcular conjunto de paréntesis que minimizan multiplicaciones escalares.

```
1: procedure MinMatMul_Backtracking(D)
2:     let M[1..|D| − 1, 1..|D| − 1] be a matrix filled with 0
3:     let B[1..|D| − 1, 1..|D| − 1] be a matrix filled with 0
4:     for i ← |D| − 2 to 1 step −1 do
5:         for j ← i + 1 to |D| − 1 do
6:             q ← ∞
7:             for k ← i to j − 1 do
8:                 v ← M[i, j] + M[k + 1, j] + D[i]D[k + 1]D[j + 1]
9:                 if v < q then
10:                    q ← v ∧ B[i, j] ← k
11:                end if
12:            end for
13:            M[i, j] = q
14:        end for
15:    end for
16:    return B
17: end procedure
```

**Algoritmo 3** Generar representación de parentización sobre las matrices.

```
1: procedure DrawParentheses(B, i, j)
2:     if i = j then
3:         return Concat("A", String(i))
4:     else
5:         k ← B[i, j]
6:         left ← DrawParentheses(B, i, k)
7:         if i ≠ k then
8:             left ← Concat("(", left, ")")
9:         end if
10:        right ← DrawParentheses(B, k + 1, j)
11:        if k + 1 ≠ j then
12:            right ← Concat("(", right, ")")
13:        end if
14:        return Concat(left, right)
15:    end if
16: end procedure
```

#### 3.2.1. Análisis de complejidad

El algoritmo de solución se divide en dos partes: encontrar las combinaciones de paréntesis que llevan a la solución óptima, y dibujar los paréntesis en una cadena de caracteres, $O = O_1 + O_2$, en donde solo se preserva la de mayor complejidad.

La complejidad de encontrar la combinación de paréntesis se puede analizar por inspección de código: al tener tres ciclos anidados, se puede expresar como $O_1(n^3)$.

Por su parte, dibujar los paréntesis es un algoritmo dividir y vencer, y su complejidad temporal se puede expresar como:

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

Aplicando teorema maestro:

1. $1 = n^{\log_b(a-\epsilon)} = n^{\log_2(2-\epsilon)} \wedge \epsilon = 1 \rightarrow \Theta(n^{\log_b a}) = \Theta(n)$
2. $1 = n^{\log_b a} \log_2^k n = n^{\log_2 2} \log_2^k n \wedge k =?$
3. $1 = n^{\log_b(a-\epsilon)} = n^{\log_2(2+\epsilon)} \wedge \epsilon = -1$

Por ende, la complejidad de dibujar los paréntesis es $\Theta_2(n)$. Debido a que $O_1 > \Theta_2$, la complejidad del algoritmo es $O(n^3)$.

# 4. Análisis experimental

En esta sección se presentarán algunos experimentos para probar el algoritmo solución presentado con distintas configuraciones de secuencias de matrices.

## 4.1. Protocolo

Para todas las configuraciones de secuencias de matrices, el protocolo es el mismo:

1. Definir dentro de un archivo CSV, cada configuración de matrices a partir de la secuencia $D$ a probar, una por cada línea.
2. Cargar el archivo en memoria
3. Por cada experimento, imprimir en pantalla la asociación de paréntesis trazada
4. Se comentarán las conclusiones necesarias por cada experimento

#### 4.1.1. Configuración de matrices iguales

Se definirá como uno de los experimentos, una configuración de matrices $D$ donde $d_i = d_{i+1} \forall i \in [1, |D|-1]$.

#### 4.1.2. Configuración de matrices ascendientes

Se definirá como uno de los experimentos, una configuración de matrices $D$ donde $d_i < d_{i+1} \forall i \in [1, |D|-1]$.

#### 4.1.3. Configuración de matrices descendientes

Se definirá como uno de los experimentos, una configuración de matrices $D$ donde $d_{i+1} < d_i \forall i \in [1, |D|-1]$.

#### 4.1.4. Configuración de matrices aleatorias

Se definirá una cantidad arbitraria de matrices sin ninguna configuración aparente.

# 5. Resultados de los experimentos

El algoritmo de solución se implementa en Julia y los experimentos se realizan siguiendo el protocolo definido en la sección 4.1.

## 5.1. Configuración de matrices iguales

Para este experimento, se define la configuración $D = \langle d_i = 100 \mid 1 \leq i \leq 10 \rangle$. Es decir, 9 matrices $\mathbb{A} = \left\langle A_i \in \mathbb{R}^{100 \times 100} \right\rangle$. El resultado del experimento es:

$$A_1(A_2(A_3(A_4(A_5(A_6(A_7(A_8A_9)))))))$$

Esta agrupación es coherente, ya que, al todas las dimensiones ser siempre iguales, el algoritmo solo va a encontrar el primer mínimo después de infinito, y nunca volver a cambiarlo. Por como está configurado el algoritmo, sencillamente coloca un paréntesis en cada $k \in [1, 9]$.

## 5.2. Configuración de matrices ascendentes

Para este experimento, se define la configuración $D = \langle d_i = d_{i-1} + 10 \mid 1 \leq i \leq 10 \wedge d_1 = 10 \rangle$. Es decir, 9 matrices $\mathbb{A} = \left\langle A_1 \in \mathbb{R}^{10 \times 20}, A_2 \in \mathbb{R}^{20 \times 30}, \cdots, A_9 \in \mathbb{R}^{90 \times 100} \right\rangle$. El resultado del experimento es:

$$(((((((A_1A_2)A_3)A_4)A_5)A_6)A_7)A_8)A_9$$

Por razonamiento lógico, se puede determinar que esta configuración es coherente. Si $r_1$ es la menor cantidad de filas, y después de cada operación $A_iA_{i+1}$ la matriz resultante $\in \mathbb{R}^{r_i \times c_{i+1}}$, tiene sentido dejar siempre la matriz de menor dimensión a la izquierda para asegurar siempre la menor cantidad de operaciones.

## 5.3. Configuración de matrices ascendentes

Para este experimento, se define la configuración $D = \langle d_i = d_{i-1} - 10 \mid 1 \leq i \leq 10 \wedge d_1 = 100 \rangle$. Es decir, 9 matrices $\mathbb{A} = \left\langle A_1 \in \mathbb{R}^{100 \times 90}, A_2 \in \mathbb{R}^{90 \times 80}, \cdots, A_9 \in \mathbb{R}^{20 \times 10} \right\rangle$. El resultado del experimento es:

$$A_1(A_2(A_3(A_4(A_5(A_6(A_7(A_8A_9)))))))$$

Por razonamiento lógico, se puede determinar que esta configuración es coherente. Si $c_9$ es la menor cantidad de columnas, y después de cada operación $A_iA_{i+1}$ la matriz resultante $\in \mathbb{R}^{r_i \times c_{i+1}}$, tiene sentido dejar siempre la matriz de menor dimensión a la derecha para asegurar siempre la menor cantidad de operaciones.

## 5.4. Configuración de matrices aleatorias

Debido a que no se tiene un mecanismo confíable para determinar si el resultado del experimento es coherente, excepto calcular a mano las multiplicaciones, sencillamente se presentan los resultados obtenidos para cada configuración:

### 5.4.1. Experimento aleatorias 1

**Entrada**

- $D = \langle 10, 100, 5, 50 \rangle$

**Salida**

- $(A_1A_2)A_3$

#### 5.4.2. Experimento aleatorias 2

**Entrada**

- $D = \langle 100, 500, 20, 100, 1000, 25, 300, 100, 1000 \rangle$

**Salida**

- $(A_1 A_2)(((((A_3 A_4) A_5) A_6) A_7) A_8)$

#### 5.4.3. Experimento aleatorias 3

**Entrada**

- $D = \langle 200, 10, 1000, 5000, 200, 10, 10000, 100, 10 \rangle$

**Salida**

- $A_1((((A_2 A_3) A_4) A_5)((A_6 A_7) A_8))$

#### 5.4.4. Experimento aleatorias 4

**Entrada**

- $D = \langle 1000, 500, 5000, 200, 3000, 2000, 10, 500, 50, 20 \rangle$

**Salida**

- $(A_1(A_2(A_3(A_4(A_5 A_6)))))((A_7 A_8) A_9)$

#### 5.4.5. Experimento aleatorias 5

**Entrada**

- $D = \langle 200, 25, 100, 200, 500, 1000, 500, 2000, 3000 \rangle$

**Salida**

- $A_1((((((A_2 A_3) A_4) A_5) A_6) A_7) A_8)$