Implementación y análisis de la multiplicación de matrices

Benjamin Alejandro Mosso Miller
Diego Alexis Salazar Jara

29 de mayo de 2025

1. Respuestas del Problema 2

1. Propiedad 1:

$$\begin{cases} C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\ C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{cases}$$

2. Propiedad 2:

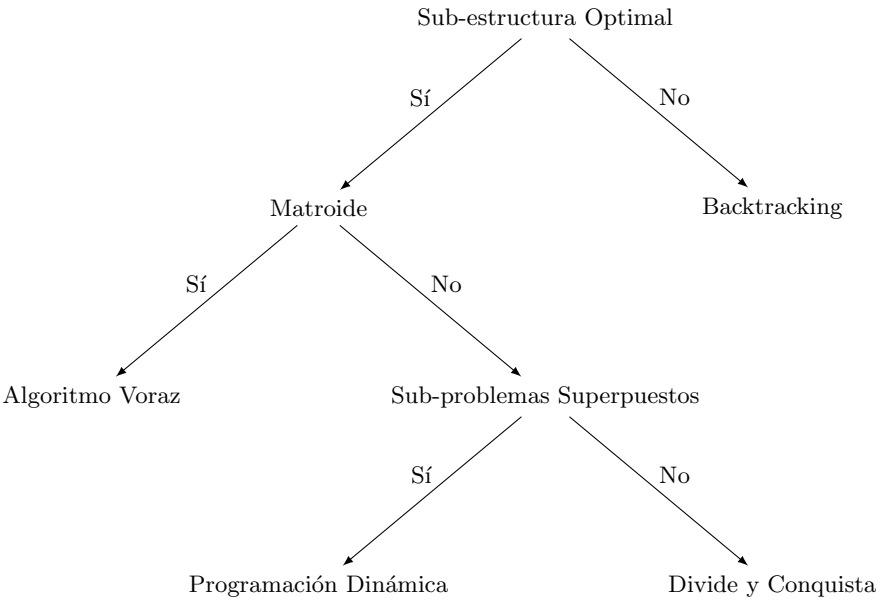
$$\begin{aligned} M &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ N &= (A_{21} + A_{22})B_{11} \\ O &= A_{11}(B_{12} - B_{22}) \\ P &= A_{22}(B_{21} - B_{11}) \\ Q &= (A_{11} + A_{12})B_{22} \\ R &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ S &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

1. Demostración del uso de técnica de diseños para la propiedad 1 y 2 mencionadas anteriormente:

   a) **Para la propiedad 1:** Respecto a la existencia de subestructura optimal, la respuesta es afirmativa, ya que la multiplicación original puede dividirse en operaciones más pequeñas que luego se combinan para obtener el resultado final. No posee matroide porque no cumple con todas las propiedades que definen a una matroide; basta con observar que $AB \neq BA$, lo que implica que el orden de multiplicación afecta el resultado. Además, no presenta subproblemas superpuestos, ya que cada producto de submatrices es único y no se reutiliza en otros cálculos.

   b) **Para la propiedad 2:** Esta fórmula corresponde al **algoritmo de Strassen**, que reduce la cantidad de multiplicaciones necesarias para matrices, buscando así disminuir el número de llamadas recursivas. Presenta subestructura optimal, tal como se explicó en el primer punto, y tampoco posee matroide por la no conmutatividad de la multiplicación ($AB \neq BA$). La diferencia principal con la **propiedad 1** radica en la reestructuración de la división y combinación, que permite reducir la cantidad de subproblemas de multiplicación recursiva.

   Por lo tanto, se concluye que ambas propiedades pertenene a la categoría de **Divide y Conquista**, como se muestra en el diagrama presentado.



2. Implementación de algoritmo tradicional, divide y conquista, como también el algoritmo de Strassen. Luego en la tabla se muestras los resultados de ejecución de cada uno de los algoritmos de una matriz n x n.

```
1   Matrix traditional_multiplication(const Matrix &A, const Matrix &B)
2   {
3       int n = A.size();
4       Matrix C(n, std::vector<int>(n, 0));
5       for (int i = 0; i < n; ++i)
6           for (int j = 0; j < n; ++j)
7               for (int k = 0; k < n; ++k)
8                   C[i][j] += A[i][k] * B[k][j];
9       return C;
10  }
```

Código 1: Algoritmo Tradicional.

```
Matrix dr1_multiplication(const Matrix &A, const Matrix &B)
{
    int n = A.size();

    if (n <= 16)
        return traditional_multiplication(A, B);

    Matrix A11, A12, A21, A22;
    Matrix B11, B12, B21, B22;

    split_matrix(A, A11, A12, A21, A22);
    split_matrix(B, B11, B12, B21, B22);

    Matrix C11 = add_matrix(dr1_multiplication(A11, B11), dr1_multiplication(A12, B21));
    Matrix C12 = add_matrix(dr1_multiplication(A11, B12), dr1_multiplication(A12, B22));
    Matrix C21 = add_matrix(dr1_multiplication(A21, B11), dr1_multiplication(A22, B21));
    Matrix C22 = add_matrix(dr1_multiplication(A21, B12), dr1_multiplication(A22, B22));

    return join_matrix(C11, C12, C21, C22);
}
```

Código 2: Algoritmo Divide y Conquista.

```
Matrix dr2_multiplication(const Matrix &A, const Matrix &B)
{
    int n = A.size();

    if (n <= 16)
        return traditional_multiplication(A, B);

    Matrix A11, A12, A21, A22;
    Matrix B11, B12, B21, B22;

    split_matrix(A, A11, A12, A21, A22);
    split_matrix(B, B11, B12, B21, B22);

    Matrix M = dr2_multiplication(add_matrix(A11, A22), add_matrix(B11, B22));
    Matrix N = dr2_multiplication(add_matrix(A21, A22), B11);
    Matrix O = dr2_multiplication(A11, subtract_matrix(B12, B22));
    Matrix P = dr2_multiplication(A22, subtract_matrix(B21, B11));
    Matrix Q = dr2_multiplication(add_matrix(A11, A12), B22);
    Matrix R = dr2_multiplication(subtract_matrix(A21, A11), add_matrix(B11, B12));
    Matrix S = dr2_multiplication(subtract_matrix(A12, A22), add_matrix(B21, B22));

    Matrix C11 = add_matrix(subtract_matrix(add_matrix(M, P), Q), S);
    Matrix C12 = add_matrix(O, Q);
    Matrix C21 = add_matrix(N, P);
    Matrix C22 = add_matrix(subtract_matrix(add_matrix(M, O), N), R);

    return join_matrix(C11, C12, C21, C22);
}
```

Código 3: Algoritmo de Strassen.

n	Algoritmo Tradicional	DR1	DR2
32	1,32 ms	1,92 ms	1,96 ms
64	10,37 ms	16,61 ms	15,85
128	81,8 ms	135,92 ms	117,99 ms
256	581,96 ms	548,38 ms	423,46 ms
512	2.713,04 ms	4.407,8 ms	3.007,3 ms
1024	23.561,08 ms	35.977,85 ms	21.234,77 ms
2048	263.011,17 ms	285.904,79 ms	150.243,44 ms
4096	N/A	N/A	N/A

Tabla 1: Comparación tiempos de ejecución de cada algoritmo.

3. Obtener al menos dos conclusiones, respecto del rendimiento de los algoritmos.

   El análisis comparativo de los algoritmos para multiplicar matrices n x n, basado en la Tabla 1 es el siguiente:

   a) El **algoritmo de Strassen** presenta un mejor rendimiento para matrices de tamaño grande, específicamente cuando la dimensión n es igual o superior a 1024 y 2048. Esto se debe a que reduce el número de multiplicaciones necesarias respecto al **algoritmo tradicional**, lo que se traduce en un tiempo de ejecución significativamente menor en estos casos, como se puede observar en la **Tabla 1**.

   b) En contraste, el **algoritmo tradicional** y el **algoritmo de divide y conquista (DR1)** muestran tiempos de ejecución muy similares entre sí. Esta cercanía en sus resultados sugiere que el método divide y conquista implementado no logra una optimización significativa frente al enfoque tradicional para las dimensiones probadas, posiblemente debido a que la reducción en el número de multiplicaciones no es tan eficiente o el overhead de la recursión y operaciones adicionales contrarrestan las ventajas.

   c) Por lo tanto, para matrices de gran tamaño, es claramente más eficiente utilizar el **algoritmo de Strassen**. Esto se debe a la mejor eficiencia computacional que este método ofrece al manejar matrices grandes, gracias a su técnica de reducir la cantidad total de multiplicaciones, lo cual reduce el tiempo de procesamiento conforme crece la dimensión de la matriz.

Como se puede observar en el gráfico, a mayor tamaño de la matriz, rinde mejor el algoritmo de Strassen en comparación con el tradicional o divide y conquista.

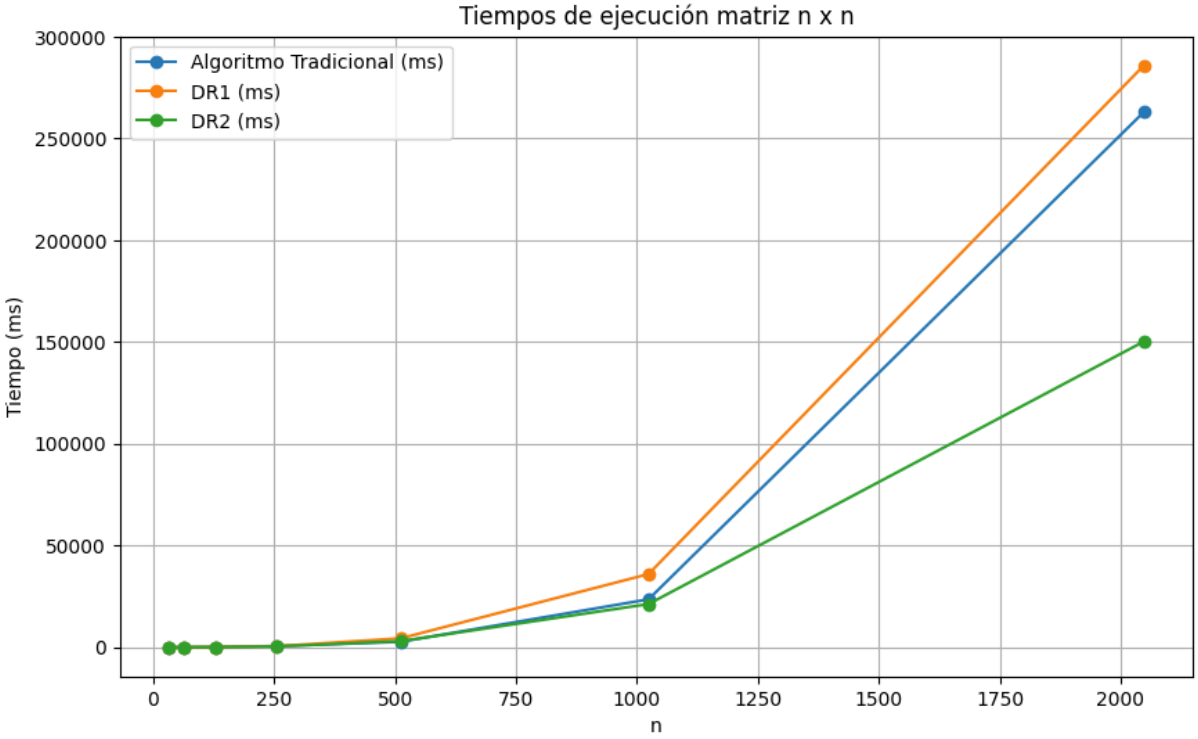


Figura 1: Comparación de rendimiento entre algoritmos de multiplicación de matrices.

4. Estudio de comportamiento asintótico para algoritmo tradicional, DR1 y DR2.

   - **Algoritmo tradicional:**

$$T(n) = n^3 \tag{1}$$

$$T(n) = \Theta(n^3) \tag{2}$$

   - **Algoritmo divide y conquista (DR1):**

$$T(n) = 8T(n/2) + \Theta(n^2) \tag{1}$$

Por Teorema Maestro Simplificado:

$$T(N) = \begin{cases} \Theta\left(n^d\right), & \text{si } a < b^d \\ \Theta\left(n^d \log n\right), & \text{si } a = b^d \\ \Theta\left(n^{\log_b a}\right), & \text{si } a > b^d \end{cases}$$

Identificar constantes:

$$a = 8, \quad b = 2, \quad d = 2 \tag{2}$$

Dado que $a > b^d$, es decir $8 > 2^2$, se cumple el tercer caso del teorema maestro simplificado, por lo que:

$$T(N) = \Theta\left(n^{log_2 8}\right) = \Theta\left(n^3\right) \tag{3}$$

- **Algoritmo de Strassen (DR2):**

$$T(n) = 7T(n/2) + \Theta(n^2) \tag{1}$$

Por Teorema Maestro Simplificado:

$$T(N) = \begin{cases} \Theta\left(n^d\right), & \text{si } a < b^d \\ \Theta\left(n^d \log n\right), & \text{si } a = b^d \\ \Theta\left(n^{\log_b a}\right), & \text{si } a > b^d \end{cases}$$

Identificar constantes:

$$a = 7, \quad b = 2, \quad d = 2 \tag{2}$$

Dado que $a > b^d$, es decir $7 > 2^2$, se cumple el tercer caso del teorema maestro simplificado, por lo que:

$$T(N) = \Theta\left(n^{log_2 7}\right) \approx \Theta\left(n^{2,8074}\right) \tag{3}$$

2. Anexos

El servidor está montado en la red universitaria de la Universidad del Bío-Bío. A continuación, se presentan las características principales del hardware y la memoria del sistema:

Procesador (CPU)	**Memoria RAM**
Intel(R) Xeon(R) CPU X5560 @ 2.80GHz	76 GiB

Tabla 2: Características principales del hardware del servidor