"Matrix Chain Multiplication" Algoritmo de Programación Dinámica

Juan Andrés Young Hoyos y Mateo Villada Higuita

1 El lenguaje de programación utilizado y por qué fue seleccionado

Usamos Python por la experiencia adquirida previamente en problemas similares, lo que facilita el desarrollo rápido y eficiente de soluciones.

2 Descripción del algoritmo. ¿Por qué funciona?

La meta es determinar el orden de multiplicación que minimiza el número de operaciones necesarias para multiplicar una secuencia de matrices.

El algoritmo funciona al dividir el problema en subproblemas más pequeños, que se resuelven de forma óptima. Cada subproblema representa el costo de multiplicar una subcadena de matrices, y el costo mínimo de cada subcadena se almacena en una tabla (`min_cost`) para evitar cálculos redundantes. La solución se construye de abajo hacia arriba en tiempo $O(n^3)$, que es más eficiente que un enfoque de fuerza bruta.

3 Descripción del mecanismo recursivo

El algoritmo utiliza una relación de recurrencia para calcular el costo mínimo de multiplicar subcadenas de matrices. Los elementos fundamentales del mecanismo recursivo son:

3.1 Variable que determina el tamaño del problema

La variable principal que determina el tamaño del problema es la longitud de la subcadena de matrices que estamos considerando en cada paso. En el código, esta longitud se representa mediante la variable `length`, que controla el tamaño de las subcadenas que se evalúan en cada iteración del bucle principal. Comenzamos con subcadenas de longitud 2 y aumentamos progresivamente hasta cubrir la cadena completa de matrices.

3.2 Casos triviales

El caso trivial ocurre cuando la subcadena contiene solo una matriz, ya que no se requiere realizar ninguna operación de multiplicación. En estos casos, el costo de multiplicación es cero, y esto se representa en la tabla `min_cost` inicializando $\texttt{min\_cost}[i][i] = 0$.

3.3 Relación de recurrencia

Para una subcadena de matrices desde A_i hasta A_j, calculamos el costo mínimo de multiplicación considerando todas las posibles posiciones k de partición. La relación de recurrencia para el costo mínimo de multiplicar matrices desde A_i hasta A_j es:

$$\texttt{min\_cost}[i][j] = \min_{i \leq k < j} (\texttt{min\_cost}[i][k] + \texttt{min\_cost}[k+1][j] + p_{i-1} \cdot p_k \cdot p_j)$$

donde p_{i-1}, p_k, y p_j representan las dimensiones de las matrices que se están multiplicando.

Esta relación permite llenar la tabla `min_cost` de manera incremental, garantizando que siempre se tengan los resultados de los subproblemas más pequeños cuando se calculan problemas más grandes.

4 Demostración matemática de por qué funciona

El algoritmo se basa en dos propiedades: la **asociatividad de la multiplicación de matrices** y la **estructura de subproblemas óptimos**. A continuación, presentamos una demostración formal que justifica el uso de la recurrencia.

4.1 Asociatividad de la multiplicación

La multiplicación de matrices es asociativa, es decir, para cualquier secuencia de matrices A, B, y C, tenemos que:

$$(A \cdot B) \cdot C = A \cdot (B \cdot C)$$

Sin embargo, el número de multiplicaciones escalares necesarias para calcular el producto completo puede variar según cómo agrupemos las matrices. Esto nos permite reorganizar las operaciones sin alterar el resultado, optimizando el número total de multiplicaciones.

4.2 Estructura de subproblemas óptimos

Para que el algoritmo sea óptimo, necesitamos que cada subproblema contenga la solución óptima para multiplicar sus propias subcadenas de matrices. Esto se cumple debido a que, para multiplicar las matrices desde A_i hasta A_j, seleccionamos un punto de partición k que minimiza el costo de multiplicación:

$$\texttt{min\_cost}[i][j] = \texttt{min\_cost}[i][k] + \texttt{min\_cost}[k+1][j] + p_{i-1} \cdot p_k \cdot p_j$$

Cada subproblema $\texttt{min\_cost}[i][k]$ y $\texttt{min\_cost}[k+1][j]$ se resuelve de forma óptima de manera independiente, y estos resultados se combinan para encontrar el costo mínimo de multiplicar A_i hasta A_j. Por lo tanto, el valor en $\texttt{min\_cost}[0][n-1]$ al final del algoritmo representa el costo mínimo para multiplicar toda la cadena de matrices.

5 Descripción de la estrategia de programación dinámica

La estrategia de programación dinámica consiste en descomponer el problema en subproblemas más pequeños, almacenar los resultados intermedios en una tabla (`min_cost`) y usar estos resultados para construir la solución óptima de forma incremental. En cada paso, calculamos el costo mínimo de multiplicar una subcadena de matrices y lo almacenamos en `min_cost`[i][j].

La tabla de particiones (`partition_indices`) también juega un papel importante, ya que registra los índices de partición óptimos para cada subcadena de matrices. Esto nos permite reconstruir la secuencia óptima de multiplicación al final, sin necesidad de recalcular los costos.

6 Funciones utilizadas

6.1 Función `matrix_chain_order`

- **Parámetros:** `dimensions` (list): Lista de dimensiones de las matrices.
- **Retorna:** Una tabla de costos mínimos (`min_cost`) y una tabla de índices de partición (`partition_indices`).
- **Descripción:** Calcula el número mínimo de multiplicaciones necesarias para multiplicar una secuencia de matrices y almacena los índices óptimos de partición.

6.2 Función `construct_optimal_order`

- **Parámetros:** `partition_indices` (list): Tabla de índices de partición, `i` (int): Índice de inicio, `j` (int): Índice de fin.
- **Retorna:** Una cadena que representa el orden óptimo de multiplicación en notación de paréntesis.
- **Descripción:** Construye recursivamente la secuencia óptima de multiplicación basada en los índices de partición almacenados en `partition_indices`.