Semana 12: Algoritmos dinámicos (III)

Multiplicación de Matrices

Producto de dos matrices

- **Dimensiones:** Si A es de dimensiones p×q y B es q×r, el producto C=A×B tiene dimensiones p×r.
- **Número de multiplicaciones requeridas:** p·q·r.

Multiplicación Encadenada

Problema

- Dada una secuencia de matrices M_1, M_2, ..., M_n con dimensiones $d_0 \times d_1$, $d_1 \times d_2$, ..., $d_{n-1} \times d_n$, determinar la forma óptima de parentetizar para minimizar el número de multiplicaciones escalares.

Propiedades

- **No conmutativa:** A×B ≠ B×A.
- **Asociativa:** (A×B)×C = A×(B×C), lo que permite diferentes formas de agrupación.

Resolución del Problema

Definición recursiva

- **Caso base:** matrices(i,i) = 0 (una sola matriz no requiere multiplicaciones).
- **Caso recursivo:** matrices(i,j) = min{i ≤ k < j} {matrices(i,k) + matrices(k+1,j) + $d_{i-1} \cdot d_k \cdot d_j$}.

Optimización mediante programación dinámica

- Construir una tabla matrices[i][j] donde cada celda almacena el costo mínimo de multiplicar las matrices desde M_i hasta M_j.

- **Recorrido por diagonales:** Calcular progresivamente las combinaciones más pequeñas hasta llegar a la solución global.

Reconstrucción de la Solución

Matriz de partición P

- Almacena el valor k que minimiza el costo en cada intervalo [i,j].
- Permite reconstruir el orden óptimo de paréntesis.

Algoritmo de reconstrucción

- Utilizar P para imprimir la secuencia óptima de multiplicaciones.

Implementación

Cálculo del costo mínimo

```
cpp
Copiar código
int multiplica_matrices(vector<int> const& D, Matriz<int> &
P) {
    int n = D.size() - 1;
    Matriz<int> matrices(n+1, n+1, 0);
    P = Matriz<int>(n+1, n+1, 0);
    for (int d = 1; d <= n-1; ++d) {
        for (int i = 1; i <= n - d; ++i) {
            int j = i + d;
            matrices[i][j] = INF;
            for (int k = i; k <= j-1; ++k) {
                int temp = matrices[i][k] + matrices[k+1]
[j] + D[i-1]*D[k]*D[j];
                if (temp < matrices[i][j]) {
                    matrices[i][j] = temp;
                    P[i][j] = k;
                }
            }
        }
    }
```

```
    return matrices[1][n];
}
```

Reconstrucción de paréntesis

```
cpp
Copiar código
void escribir_paren(int i, int j, Matriz<int> const& P) {
    if (i == j) cout << "M" << i;
    else {
        int k = P[i][j];
        cout << "(";
        escribir_paren(i, k, P);
        cout << ")*(";
        escribir_paren(k+1, j, P);
        cout << ")";
    }
}
```

Ejemplo

Dimensiones

- M_1: 10×20, M_2: 20×30, M_3: 30×40.

Solución óptima

- **Tabla matrices[i][j]:** Calcula el costo mínimo.
- **Tabla P[i][j]:** Registra los puntos de partición.
- **Resultado:** $((M_1 \times M_2) \times M_3)$ con el mínimo número de multiplicaciones.