Dynamic Programming

Chain Matrix Multiplication

**Problem.** Given $N$ matrices $A_1, \ldots, A_N$ with dimensions $A_1[m_1, m_2], A_2[m_2, m_3], \ldots, A_N[m_N, m_{N+1}]$, find the fully parenthesized product of $A_1, \ldots, A_N$ with the lowest computation cost.

**Cost of matrix product.** Given two matrices $A[m, n]$ and $B[n, k]$, their product $C[m, k] = A \times B$ is computed as follows:

$$C_{ij} = \sum_{t=1}^{n} A_{it} \cdot B_{tj}.$$  

It takes $n$ multiplications\(^1\) to compute one element of the product matrix.

There are $m \cdot k$ elements in the product matrix.

The cost of matrix product is represented in terms of number of multiplication operations. These operations are more expensive than other operations used in the computation.

The cost of a product of two matrices is $cost(A \times B) = m \cdot n \cdot k$.

**Total Number of Parenthizations.**

A naïve algorithm for solving Chain Matrix Multiplication problem is shown in Figure 1.

This algorithm is *efficient* if the total number of parenthizations is small (bounded by a polynomial).

---

\(^1\)This assumes the direct way of computing a product of two matrices. A more computationally efficient algorithm exists, but it is not usually used in practice.
Figure 1: Naïve algorithm for solving the Chain Matrix Multiplication problem.

Computing the total number of parenthizations. Let $P(n)$ be the number of possible parenthizations of $n$ matrices. We observe:

- $P(1) = 1$.
- A complete parenthization of $n$ matrices splits the matrices at some point between $k$th and $k + 1$st matrices of the input. There are $n - 1$ possible splits: between $A_1$ and $A_2$; between $A_2$ and $A_3$, . . . , between $A_{n-1}$ and $A_n$. Each of the two split parts is, in turn a complete parenthization.
- The total number of parenthizations of the form $(A_1 \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_n)$ is $P(k) \cdot P(n - k)$.
- The total number of parenthizations of a product of $n$ matrices (for $n > 1$) is then
  
  $$P(n) = \sum_{k=1}^{n-1} n - 1 P(k) \cdot P(n - k).$$

- The solution to this recurrence equation is $\Omega(2^n)$.
- This means, that the naïve algorithm in not applicable in practice.

Dynamic Programming Algorithm for Chain Matrix Multiplication

Solution Idea. For each subsequence $A_i \ldots A_j$ of matrices find the best possible parenthization.

We can do it efficiently in a bottom-up fashion because:

Optimal substructure property. Optimal substructure property is present in this problem.

If $(A_1 \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_N)$ is an optimal solution for $A_1 \times \ldots \times A_N$, then the parenthizations of $A_1 \times \ldots \times A_k$ and $A_{k+1} \times \ldots \times A_N$ must be optimal (otherwise, we can use optimal parenthizations to get a better cost estimate).

Overlapping subproblems. A lot of subproblems will overlap. E.g., $A_i \times \ldots \times A_j$ and $A_{i+1} \times \ldots \times A_{j+1}$ both share the subproblem for $A_{i+1} \times \ldots \times A_j$. 
**Algorithm MatrixChain**(N, A[1..N+1])
// A[1..N+1] is an array of matrix dimensions
begin
  m[1..N][1..N];
  s[1..N][1..N];
  // Initialize the diagonal of m
  for i ← 1 to N
    m[i, i] ← 0;
  endfor
  for l ← 2 to N do // l is length of chain
    for i ← 1 to n - l + 1 // i is start of chain
      j ← i + l - 1; // j is end of chain
      m[i, j] = −∞;
      for k ← i to j - 1 // update the score in m[i,j]
        if q < m[i, j] then
          m[i, j] = q;
          s[i, j] = k;
        endif
      endfor
    endfor
  endfor
  return m and s;
end

Figure 2: **Algorithm MatrixChain** for solving the Matrix Chain Multiplication problem.

**Data Structures.** Our algorithm will maintain two data structures:

1. **array** m[1..N, 1..N]: m[i, j] stores the information about the cheapest cost of multiplying the sequence A_i, ..., A_j.
   In the algorithm, only the top portion of this array is used (i.e., only for i ≤ j)

2. **array** s[1..N, 1..N], which allows us to construct the optimal solution.
   s[i, j] = k for i < j means that the optimal solution of subproblem A_i × ... × A_j splits the sequence at matrix A_k:
   
   \[(A_i \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_j)\]

   Note, that s will be defined only for i < j and that i ≤ s[i, j] < j.

The bottom-up dynamic programming algorithm is shown in Figure 2.