Dynamic Programming: Matrix-Chain Multiplication

Yufei Tao’s Teaching Team

Department of Computer Science and Engineering
Chinese University of Hong Kong
Matrix-Chain Multiplication

You are given an algorithm $\mathcal{A}$ that, given an $a \times b$ matrix $A$ and a $b \times c$ matrix $B$, can calculate $AB$ in $O(abc)$ time. You need to use $\mathcal{A}$ to calculate the product of $A_1A_2...A_n$ where $A_i$ is an $a_i \times b_i$ matrix for $i \in [1, n]$. This implies that $b_{i-1} = a_i$ for $i \in [2, n]$, and the final result is an $a_1 \times b_n$ matrix.

A trivial strategy is to apply $\mathcal{A}$ to evaluate the product from left to right. However, we may be able to reduce the cost by following a different multiplication order.
Example

Consider $A_1 A_2 A_3$ where $A_1$ and $A_2$ are $m \times m$ matrices, but $A_3$ is $m \times 1$.

There are two multiplication orders:

1. $(A_1 A_2) A_3$.
   The cost of computing $B = A_1 A_2$ is $O(m \cdot m \cdot m) = O(m^3)$ and $B$ is an $m \times m$ matrix. The cost of $BA_3$ is $O(m \cdot m \cdot 1) = O(m^2)$. The total cost is $O(m^3)$.

2. $A_1 (A_2 A_3)$.
   The cost of computing $B = A_2 A_3$ is $O(m \cdot m \cdot 1) = O(m^2)$ and $B$ is an $m \times 1$ matrix. The cost of $A_1 B$ is $O(m \cdot m \cdot 1) = O(m^2)$. The total cost is $O(m^2)$. 
Parenthesizing $A_1 A_2 \ldots A_n$ at $A_k$ for some $k \in [1, n - 1]$ converts the expression to $(A_1 \ldots A_k)(A_{k+1} \ldots A_n)$, after which you can parenthesize each of $A_1 \ldots A_i$ and $A_{i+1} \ldots A_n$ recursively.

A **fully parenthesized product** is

- either a single matrix or
- the product of two fully parenthesized products.

For example, if $n = 4$, then $(A_1 A_2)(A_3 A_4)$ and $((A_1 A_2)A_3)A_4$ are fully parenthesized, but $A_1(A_2 A_3 A_4)$ is not.

A fully parenthesized product determines a multiplication order that, in turn, determines the computation cost.

**Goal:** Design an algorithm to find in $O(n^3)$ time a fully parenthesized product with the smallest cost.
Recursive Structure

By parenthesizing at $A_k$, we obtain

$$\underbrace{(A_1\ldots A_k)}_{B_1} \underbrace{(A_{k+1}\ldots A_n)}_{B_2},$$

where $B_1$ is an $a_1 \times b_k$ matrix and $B_2$ is an $a_{k+1} \times b_n$ matrix.

The total cost is

$$\text{cost of computing } B_1 + \text{cost of computing } B_2 + O(a_1 b_k b_n).$$
We define $cost(i, j)$, where $1 \leq i \leq j \leq n$, to be the smallest achievable cost for calculating $A_i \ldots A_j$. Our objective is to calculate $cost(1, n)$.

If we parenthesize $A_i \ldots A_j$ at $A_k$, we obtain

$$\underbrace{(A_i \ldots A_k)}_{cost(i, k)} \underbrace{(A_{k+1} \ldots A_j)}_{cost(k+1, j)}.$$ 

The total cost is

$$cost(i, k) + cost(k + 1, j) + O(a_i b_k b_j).$$
To attain $cost(i, j)$, we should try all possible parenthesizations of $A_i \ldots A_j$. This implies:

$$
cost(i, j) = \\
\begin{cases} 
O(1) & \text{if } i = j \\
\min_{k=i}^{j-1} (cost(i, k) + cost(k + 1, j) + O(a_i b_k b_j)) & \text{if } i < j
\end{cases}
$$

By dyn. programming, we can compute $cost(1, n)$ in $O(n^3)$ time.
Consider $A_1 A_2 A_3 A_4$ where $A_1$ and $A_2$ are $m \times m$ matrices, $A_3$ is $m \times 1$, and $A_4$ is $1 \times m$.

$$
\begin{array}{cccc}
  & 1 & 2 & 3 & 4 \\
 1 &   &   &   &   \\
 2 & 0 &   &   &   \\
 3 & 0 & 0 &   &   \\
 4 & 0 & 0 & 0 &   \\
\end{array}
$$

Dynamic Programming: Matrix-Chain Multiplication
After solving all subproblems, we obtain:

Next, we apply the “piggyback technique” to generate an optimal parenthesization.
Define \( bestSub(i, j) = \)

- \( \text{nil, if } i = j; \)
- \( k, \text{ if the best parenthesization for } A_i A_{i+1} \ldots A_j \text{ is } (A_i \ldots A_k) (A_{k+1} \ldots A_j). \)

After \( cost(i, j) \) is ready for all \( i, j \), we can compute all \( bestSub(i, j) \) in \( O(n^3) \) time.
**Example:**

bestSub(1, 4) = 3, i.e., the best way to calculate $A_1 A_2 A_3 A_4$ is $(A_1 A_2 A_3) A_4$.

Similarly, bestSub(1, 3) = 1, i.e., the best way to calculate $A_1 A_2 A_3$ is $A_1 (A_2 A_3)$.

Therefore, an optimal fully parenthesized product of $A_1 A_2 A_3 A_4$ is $(A_1 (A_2 A_3)) A_4$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$O(1)$</td>
<td>$O(m^3)$</td>
<td>$O(m^2)$</td>
<td>$O(m^2)$</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>$O(1)$</td>
<td>$O(m^2)$</td>
<td>$O(m^2)$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>$O(1)$</td>
<td>$O(m^2)$</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

$A_1$: $m \times m$

$A_2$: $m \times m$

$A_3$: $m \times 1$

$A_4$: $1 \times m$