# CIS 575. Introduction to Algorithm Analysis
# Material for March 25, 2024

## Chained Matrix Multiplication



The topic of this note is presented in *Cormen*'s Section 14.2.

## 1 Chained Matrix Multiplication: Problem

In this note we shall address matrix multiplication. You may ask: didn't we do that quite recently? We did indeed, in that we discussed how to best multiply *two big square* matrices. The current focus is quite different: we shall discuss how to best multiply **an arbitrary number** of matrices which may have **any size** and **any shape**. To be specific, given a chain of matrices to be multiplied, we shall find an optimal order to do the matrix multiplications; one that involves the fewest number of multiplications of individual matrix elements (which we shall assume are integers). This problem is trivial if

- there are only two matrices to multiply (as then only one order is possible), or
- the matrices are all square matrices (as then all orders require the same number of integer multiplications).

We shall implicitly assume that the matrices are not so huge that it will pay off to apply sophisticated techniques such as Strassen's algorithm.

**Matrix Multiplication** We may know from algebra that the **matrix product** $A\ B$ is well-defined iff there exists natural numbers $m, n, p$ such that

- $A$ is an $m \times n$ matrix
- $B$ is an $n \times p$ matrix

in which case $A\ B$ is an $m \times p$ matrix $C$ where $c_{ij}$, the element of $C$ in the $i$'th row (from top) and the $j$'th column (from left), is given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

For example, with $m = 2$ and $n = 4$ and $p = 3$, we have

$$\begin{pmatrix} 1 & 3 & 5 & 8 \\ 4 & 2 & 7 & 3 \end{pmatrix} \begin{pmatrix} 2 & 2 & 4 \\ 1 & 0 & 7 \\ 0 & 5 & 3 \\ 8 & 1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1\cdot 2+3\cdot 1+5\cdot 0+8\cdot 8 & 1\cdot 2+3\cdot 0+5\cdot 5+8\cdot 1 & 1\cdot 4+3\cdot 7+5\cdot 3+8\cdot 0 \\ 4\cdot 2+2\cdot 1+7\cdot 0+3\cdot 8 & 4\cdot 2+2\cdot 0+7\cdot 5+3\cdot 1 & 4\cdot 4+2\cdot 7+7\cdot 3+3\cdot 0 \end{pmatrix}$$

$$= \begin{pmatrix} 69 & 35 & 40 \\ 34 & 46 & 51 \end{pmatrix}$$

where the calculation involves $2\cdot 3\cdot 4$ integer multiplications. In general, the product can be calculated by the algorithm

```
for i ← 1 to m
    for j ← 1 to p
        c_ij ← 0
        for k ← 1 to n
            c_ij ← c_ij + a_ik × b_kj
```

which carries out **mnp** integer multiplications.

**Chains** Now assume, for $A$ an $m \times n$ matrix and $B$ an $n \times p$ matrix and $C$ a $p \times q$ matrix, that we must compute

$$A\ B\ C$$

Since matrix multiplication is **associative**, we may compute that either as

1. $(A\ B)\ C$, or
2. $A\ (B\ C)$.

On the other hand, as matrix multiplication is **not communitative**, we may **not** compute it as say $(B\ A)\ C$ (which is not even well-defined unless $m = n = p$).

While 1 and 2 above are equivalent in that they yield the same final outcome, they may entail quite different costs:

option 1 requires $mnp + mpq$ integer multiplications

option 2 requires $npq + mnq$ integer multiplications.

If say $m = p = 100$ and $n = q = 2$ then option 1 will require 40,000 integer multiplications but option 2 only 800 integer multiplications!

We should now be convinced that it may make a tremendous difference in which order one multiplies the matrices. It is therefore worth to spend some effort to come up with a good (even optimal) order.

One could hope that there is some simple ("greedy") strategy to pick the best sequence. For example, to do the cheapest multiplication first. But now assume that $m = 3$, $n = 2$, $p = 6$ and $q = 4$; then $AB$ (36 integer multiplications) is cheaper than $BC$ (48 integer multiplications) but

option 1 (which first computes $AB$) requires $36+72 = 108$ integer multiplications

option 2 (which first computes $BC$) requires $48+24 = 72$ integer multiplications.

You may try other strategies, but it's very unlikely that you'll find any strategy that *always* produces an optimal order.

Therefore we shall go for a solution based on Dynamic Programming. Let us first state the general problem we want to solve.

**Goal** Given a **list of dimensions**, we want to **minimize** the number of integer multiplications required to multiply a chain of matrices with those dimensions. That is, we are given numbers $d_0, d_1, \ldots, d_n$ (with $n \geq 1$) and want to find the smallest number of integer multiplications needed to compute $A_1 \ \ldots \ A_n$ where for each $i \in 1 \ldots n$, $A_i$ is a $d_{i-1} \times d_i$ matrix. Note that this number depends only on the dimensions, not on the particular arrays with those dimensions.

# 2 Chained Matrix Multiplication: Solution

As usual, a key step is to settle on the subproblems to be solved. We shall build a table $M$ such that for all $i, j$ with $1 \leq i \leq j \leq n$,

> $M[i, j]$ is the minimum number of integer multiplications needed to compute $A_i \ \ldots \ A_j$ (a $d_{i-1} \times d_j$ matrix).

The desired answer can thus be found as $M[1, n]$.

**Basic Equations** Since it does not require any multiplication to compute $A_i$, we have

$$M[i, i] = 0 \tag{1}$$

Since $A_i$ has dimensions $d_{i-1} \times d_i$, and $A_{i+1}$ has dimensions $d_i \times d_{i+1}$, for $i \in 1 \ldots n-1$ we get

$$M[i, i+1] = d_{i-1} d_i d_{i+1} \tag{2}$$

**Recursive Equation** Now assume that we must compute $A_i \ \ldots \ A_j$ where $i < j$. There exists $k$ such that we

1. first compute $A_i \ \ldots \ A_k$, which can be done using $M[i, k]$ integer multiplications; this results in a $d_{i-1} \times d_k$ matrix $B_1$
2. next compute $A_{k+1} \ \ldots \ A_j$, which can be done using $M[k+1, j]$ integers multiplications; this results in a $d_k \times d_j$ matrix $B_2$
3. finally multiply $B_1$ and $B_2$, which can be done using $d_{i-1} \cdot d_k \cdot d_j$ integer multiplications.

Here $k$ must be at least $i$ and at most $j-1$. We infer that that we can find $M[i, j]$ as

$$M[i, j] = \min_{k \in i \ldots j-1} (M[i, k] + M[k+1, j] + d_{i-1} \cdot d_k \cdot d_j) \tag{3}$$

When $j = i + 1$, only one value of $k$ is possible: $k = i$; then Equation (3) becomes

$$M[i, i+1] = M[i, i] + M[i+1, i+1] + d_{i-1} \cdot d_i \cdot d_{i+1} = d_{i-1} \cdot d_i \cdot d_{i+1}$$

which shows that Equation (2) is redundant as it is a special case of Equation (3).

**Implementation** When implementing the equations, as always one has to be careful about computing the entries in the right order: never refer to an entry not yet computed. One proper approach is:

```
for i ← n downto 1
    M[i,i] ← 0
    for j ← i + 1 to n
        // compute M[i,j]
        M[i,j] ← ∞
        for k ← i to j − 1
            v ← M[i,k] + M[k+1,j] + d_{i−1} × d_k × d_j
            if v < M[i,j]
                M[i,j] ← v
```

In order to find not just the minimum number of integer multiplications, but also to be able to retrieve an optimal order of matrix multiplications, we should for each $i, j$ record the best $k$.

**Example** Let us return to our previous example which had $m = 3$, $n = 2$, $p = 6$ and $q = 4$; in our general setting this will translate to $n = 3$ and $d_0 = 3$, $d_1 = 2$, $d_2 = 6$, and $d_3 = 4$. The algorithm written above will successively compute:

$$\begin{aligned} M[3,3] &= 0 \\ M[2,2] &= 0 \\ M[2,3] &= M[2,2] + M[3,3] + 2 \cdot 6 \cdot 4 = \mathbf{48} \\ M[1,1] &= 0 \\ M[1,2] &= M[1,1] + M[2,2] + 3 \cdot 2 \cdot 6 = \mathbf{36} \end{aligned}$$

before we compute the desired answer:

$$\begin{aligned} M[1,3] &= \min(M[1,1] + M[2,3] + d_0 \cdot d_1 \cdot d_3,\ M[1,2] + M[3,3] + d_0 \cdot d_2 \cdot d_3) \\ &= \min(48 + 24,\ 36 + 72) = \mathbf{72} \end{aligned}$$

**Space Use** It is easy to see that the algorithm uses space in $\Theta(n^2)$.

**Running Time** Unlike previous examples of dynamic programming, each entry can *not* be computed in constant time. The asymptotic running time is given by the summation

$$\sum_{i=1}^{n} \sum_{j=i}^{n} \Theta(j - i + 1)$$

which (using *Howell*'s Theorem 3.28) we can simplify into

$$\sum_{i=1}^{n} \sum_{q=1}^{n-i+1} \Theta(q) = \sum_{i=1}^{n} \Theta((n - i + 1)^2) = \sum_{i=1}^{n} \Theta(i^2) \in \mathbf{\Theta(n^3)}$$

Thus the optimal order of doing the matrix multiplications can be found in **cubic** time of the number of matrices to be multiplied. Observe that this is the running time of the *scheduling*, not the running time of the multiplications themselves. But hopefully the time saved due to the optimal evaluation order will exceed the time used on computing that order!