Multiplying a Sequence of Matrices

Suppose we want to multiply a long sequence of matrices $A \times B \times C \times D \ldots$.

Multiplying an $X \times Y$ matrix by a $Y \times Z$ matrix (using the common algorithm) takes $X \times Y \times Z$ multiplications.

$$\begin{bmatrix} 2 & 3 \\ 3 & 4 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix} \qquad \begin{bmatrix} 13 & 18 & 23 \\ 18 & 25 & 32 \\ 23 & 32 & 41 \end{bmatrix}$$

We would like to avoid big intermediate matrices, and since matrix multiplication is *associative*, we can parenthesise however we want.

Matrix multiplication is *not communitive*, so we cannot permute the order of the matrices without changing the result.

Example

Consider $A \times B \times C \times D$, where A is 30×1, B is 1×40, C is 40×10, and D is 10×25.

There are three possible parenthesizations:

$$((AB)C)D = 30{\times}1{\times}40{+}30{\times}40{\times}10{+}30{\times}10{\times}25 = 20,700$$

$$(AB)(CD) = 30{\times}1{\times}10{+}40{\times}10{\times}25{+}30{\times}40{\times}25 = 41,200$$

$$A((BC)D) = 1{\times}40{\times}10{+}1{\times}10{\times}25{+}30{\times}1{\times}25 = 1400$$

The order makes a big difference in real computation. How do we find the best order?

Let $M(i,j)$ be the *minimum* number of multiplications necessary to compute $\prod_{k=i}^{j} A_k$.

The key observations are

- The outermost parentheses partition the chain of matricies (i,j) at some k.
- The optimal parenthesization order has optimal ordering on either side of k.

A recurrence for this is:

$$
\begin{aligned}
M(i,j) &= Min_{i \le k \le j-1}[M(i,k) + M(k+1,j) + d_{i-1}d_kd_j] \\
M(i,j) &= 0
\end{aligned}
$$

If there are n matrices, there are $n+1$ dimensions.

A direct recursive implementation of this will be exponential, since there is a lot of duplicated work as in the Fibonacci recurrence.

Divide-and-conquer is seems efficient because there is no overlap, but . . .

There are only $\binom{n}{2}$ substrings between 1 and n. Thus it requires only $\Theta(n^2)$ space to store the optimal cost for each of them.

We can represent all the possibilities in a triangle matrix:

SHOW THE DIAGONAL MATRIX

We can also store the value of k in another triangle matrix to reconstruct to order of the optimal parenthesisation.

The diagonal moves up to the right as the computation progresses. On each element of the kth diagonal $|j-i| = k$.

For the previous example:

SHOW BIG FIGURE OF THE MATRIX

```
Procedure MatrixOrder
for i = 1 to n do M[i,j] = 0
for diagonal = 1 to n − 1
        for i = 1 to n − diagonal do
                j = i + diagonal
                M[i,j] = min_{i=k}^{j−1} [M[i,k] + M[k+1,j] + d_{i−1} d_k d_j]
                faster(i,j) = k
return [m(1,n)]
```

```
Procedure ShowOrder(i,j)
if (i = j) write (A_i)
else
        k =factor(i,j)
        write "("
        ShowOrder(i,k)
        write "*"
        ShowOrder (k+1,j)
        write ")"
```

A dynamic programming solution has three components:

1. Formulate the answer as a recurrence relation or recursive algorithm.

2. Show that the number of different instances of your recurrence is bounded by a polynomial.

3. Specify an order of evaluation for the recurrence so you always have what you need.