DYNAMIC PROGRAMMING
MATRIX CHAIN MULTIPLICATION

DR. GODFREY MUGANDA

1. Review of Matrix Multiplication

A matrix with m rows and n columns is said to be an $m \times n$ matrix.

Given two matrices A and B, the matrix product $A \times B$ can only be formed if the number of columns of A is equal to the number of rows of B. That is, A should be an $m \times n$ matrix and B should be an $n \times p$ matrix for some m and p.

The product matrix $P = A \times B$ will have m rows and p columns. Thus it will have the same number of rows as A (the first matrix) and the same number of columns as B (the second matrix).

To compute the element P_{ij} in row i and column j, you multiply row i of A with column j of B:

$$\begin{bmatrix} \vdots & \vdots & \cdots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \vdots & \vdots & \cdots & \vdots \end{bmatrix} \begin{bmatrix} \cdots & b_{1j} & \cdots \\ \cdots & b_{2j} & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & b_{nj} & \cdots \end{bmatrix}$$

so we get

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

Thus one entry of the product matrix can be computed using $\mathrm{O}(n)$ multiplications. Because P has $m \times p$ entries, the product matrix can be computed using mnp multiplications.

In some applications, many different sequences of matrices with the same fixed dimensions need to be multiplied. Thus, for fixed dimensions

$$m_0, m_1, \ldots, m_n$$

there are many sequences of matrices

$$\begin{array}{cccc} A_1 & A_2 & \cdots & A_n \\ m_0 \times m_1 & m_1 \times m_2 & & m_{n-1} \times m_n \end{array} \tag{1}$$

The dimensions of the matrices stay fixed and only the values of the matrix entries vary. An example of this is weather prediction, where the matrix entries could represent readings on different days.

Now matrix multiplication is *associative* but not *commutative*. Not commutative means that in general, $A \times B \neq B \times A$: that is, left-to-right order of multiplication matters.

By associative, we mean that given a sequence of three matrices A, B, and C to be multiplied, you can group them two at a time any way you please, without affecting the product:

$$A \times (B \times C) = (A \times B) \times C$$

On pages 168 and 169, the text book shows that for four matrices A, B, C, and D with respective dimensions

$$50 \times 20, \quad 20 \times 1, \quad 1 \times 10 \quad \text{and} \quad 10 \times 100, \tag{2}$$

the grouping $A\times((B\times C)\times D)$ requires 120,200 multiplications whereas the grouping $(A \times B) \times (C \times D)$ requires only 7,000 multiplications. That is a big difference.

Exercise: With the dimensions of (2), how many multiplications are required to compute the products

(1) $A \times (B \times C)$?
(2) $(A \times B) \times C)$?
(3) $(B \times C) \times D$?

If you are going to multiply many sequences of matrices with the same dimensions, it makes sense to figure out once and for all the optimal grouping. This is where dynamic programming comes in.

If we are trying to multiply the sequence (1), the first decision is to make a top-level grouping. Let $k = 1, \ldots, n-1$. The we make a top level grouping as follows

$$A_1, A_2, \ldots A_k, \quad A_{k+1}, \ldots A_n \tag{3}$$

and then from there, we go on to recursively determine the optimal way to multiply the sequences

$$A_1, A_2, \ldots A_k \tag{4}$$

and

$$A_{k+1}, \ldots A_n \tag{5}$$

It soon becomes clear that the general subproblem is to find the optimal way to multiply the contiguous subsequence

$$A_i, A_2, \ldots A_j \quad \text{for } 1 \le i \le j \le n. \tag{6}$$

Let $C(i, j)$ be the optimal number of multiplications for (6). As before, we select a split point k such that $i \le k < j$. This will result in two subproblems

$$A_i, A_2, \ldots A_k \tag{7}$$

with optimal number of multiplications $C(i, k)$, and

$$A_{k+1}, \ldots A_j \tag{8}$$

with optimal number of multiplications and $C(k + 1, j)$.

Whenever you multiply a sequence of matrices,

$$\underset{m_{i-1} \times m_i}{A_i} \quad \cdots \times \cdots \quad \underset{m_{k-1} \times m_k}{A_k} \tag{9}$$

All the middle dimensions drop out and the product has dimensions $m_{i-1} \times m_k$: it has the same number of rows as the first matrix and the same number of columns as the last matrix.

This means that the product of (7) has dimensions $m_{i-1} \times m_k$, and the product of (8) has dimensions $m_k \times m_j$. We must multiply those two matrices to get the product of (6), and this multiplication costs $m_{i-1} \times m_k \times m_j$.

What this means is that once we commit to a split point k to split the sequence (6), the best we can do is

$$C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j. \tag{10}$$

We have no way of knowing ahead of time which split point k will give the best performance, so we must consider them all and select the k that achieves the minimum. We get the recurrence relation

$$C(i,j) = \min_{i \le k < j} \{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j\} \tag{11}$$

We can record the optimal split point with

$$S(i,j) = \text{ the } k \text{ that achieves the minimum in (11).} \tag{12}$$

For the boundary conditions, note that the recurrence pushes i and j closer together. The boundary condition is when $i = j$, and you have a sequence consisting of a single matrix A_i. In this case,

$$C(i,j) = C(i,i) = 0 \quad \text{for all } i \text{ with } 1 \le i \le n. \tag{13}$$

Now we need to set up the solution tables $C[1 \ldots n, 1 \ldots n]$ and $S[1 \ldots n.1 \ldots n]$.

The values $C[i,i]$ that correspond to the boundary conditions are on the main diagonal:

$$\begin{bmatrix} 0 & & & & \\ & 0 & & & \\ & & 0 & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix} \tag{14}$$

Because $i < j$, we are only interested in $C[i,j]$ in the half of the matrix that lies above the main diagonal. The part below the main diagonal is not used.

Looking at (11), we can tell that we should fill in the values $C[i,j]$ in increasing order of $j - i$, that is, in increasing order of diagonals.

If we number the diagonals $s = 0, 1, \ldots n-1$, where diagonal $s = 0$ is the main diagonal, we have something like this when $n = 5$ and we fill each entry is diagonal s with the number s:

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ & 0 & 1 & 2 & 3 \\ & & 0 & 1 & 2 \\ & & & 0 & 1 \\ & & & & 0 \end{bmatrix} \tag{15}$$

Notice in particular that in diagonal s, the difference $j - i$ is always equal to s.

The algorithm will start by filling in the main diagonal, and then compute all values in diagonal $s = 1$ by rows, then all values in diagonal $s = 2$, and so on through the last diagonal $s = n - 1$.

Notice also that in diagonal s, the rows start at 1 and end at $n - s$. Remember that in row i of diagonal s, the column j is $j = i + s$.

Put it all together, and we have

for $i = 1$ to n $C[i, i] = 0$ end for
for $s = 1$ to $n - 1$ // proceed by diagonals s
  for $i = 1$ to $n - s$ // rows i within diagonal s
    $j = i + s$
    $C(i, j) = \min_{i \le k < j}\{C(i, k) + C(k + 1, j) + m_{i-1} \cdot m_k \cdot m_j\}$
  end for
end for

Notice that it takes $\mathrm{O}(j - i)$, which is $\mathrm{O}(n)$, to evaluate equation (11). there are $\mathrm{O}(n^2)$ entries in the table, each of which takes $\mathrm{O}(n)$ time to fill. Therefore this algorithm runs in $\mathrm{O}(n^3)$.

How bad would a brute force algorithm do? It turns out that there exponentially many ways to group a sequence of matrices so you can multiply, so a brute force algorithm, say based on backtracking, would require exponential time.

Exercise: Modify this algorithm so that you replace the expression (11) with a loop that computes $C[i, j]$, and stores the optimal split point k in a split table $S[i, j]$.

Exercise: Given such a split table $S[1 \ldots n, 1 \ldots n]$, write a method that prints out the full parenthesization that represents the optimal grouping for the purpose of multiplication. For example, if $n = 3$, your method might print out

$$((A_1)(A_2A_3)).$$

It is ok to use extra parentheses.

Example: Let us work out the algorithm by hand for $n = 4$ and the four matrices for four matrices A, B, C, and D with respective dimensions

$$50 \times 20, \quad 20 \times 1, \quad 1 \times 10 \quad \text{and} \quad 10 \times 100, \tag{16}$$

and build the solution tables. Follow up by writing out the optimal parenthesization.

Note this is the same problem that appears in the class text book at the bottom of page 169. The data for the problem is the list of dimensions

m_0	m_1	m_2	m_3	m_4
50	20	1	10	100

We need two solution tables, a cost table $C[1 \ldots 4, 1 \ldots 4]$ and a split point table $S[1 \ldots 4, 1 \ldots 4]$. In our example, we will represent the split point as subscript in the cost table.

Executing the initialization code sets the main diagonal elements $C[i, i]$ to all zeros:

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

Now let us look at diagonal 1 when $s = 1$. There are 3 elements in this diagonal,$C[1, 2]$, $C[2, 3]$ and $C[3, 4]$. For each of these, there is only one possible value for the split point.

$$C[1,2] = C[1,1] + C[2,2] + m_0 m_1 m_2 = 0 + 0 + m_0 m_1 m_2 = 50 \times 20 \times 1 = 1000$$
$$C[2,3] = C[2,2] + C[3,3] + m_1 m_2 m_3 = 0 + 0 + m_1 m_2 m_3 = 20 \times 1 \times 10 = 200$$
$$C[3,4] = C[3,3] + C[4,4] + m_2 m_3 m_4 = 0 + 0 + m_2 m_3 m_4 = 1 \times 10 \times 100 = 1000$$

The solution tables now look like this:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1000_1 & & \\ & 0 & 200_2 & \\ & & 0 & 1000_3 \\ & & & 0 \end{bmatrix} \end{array}$$

Moving on to the next diagonal, we need to compute $C[1, 3]$ and $C[2, 4]$. For each of these entries, we have 2 choices for the split point:

$$C[1,3] = \min \begin{cases} k = 1: & C[1,1] + C[2,3] + m_0 m_1 m_3 = 0 + 200 + 50(20)(10) = 10200 \\ k = 2: & C[1,2] + C[3,3] + m_0 m_2 m_3 = 1000 + 0 + 50(1)(10) = 1500 \end{cases}$$

so that the minimum cost is 1500 that occurs at split point $k = 2$.

$$C[2,4] = \min \begin{cases} k = 2: & C[2,2] + C[3,4] + m_1 m_2 m_4 = 0 + 1000 + 20(1)(100) = 3000 \\ k = 3: & C[2,3] + C[4,4] + m_1 m_3 m_4 = 200 + 0 + 20(10)(100) = 20200 \end{cases}$$

so this minimum cost is 3000 with split point $k = 2$.

The solution tables now look like this:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1000_1 & 1500_2 & \\ & 0 & 200_2 & 3000_2 \\ & & 0 & 1000_3 \\ & & & 0 \end{bmatrix} \end{array}$$

And finally,

$$C[1,4] = \min \begin{cases} k = 1: & C[1,1] + C[2,4] + m_0 m_1 m_4 = 0 + 3000 + 50(20)(100) = 103000 \\ k = 2: & C[1,2] + C[3,4] + m_0 m_2 m_4 = 1000 + 1000 + 50(1)(100) = 7000 \\ k = 3: & C[1,3] + C[4,4] + m_0 m_3 m_4 = 1500 + 0 + 50(10)(100) = 51500 \end{cases}$$

Thus the minimum value is 7000 with split point $k = 2$.

Filling in this information in the solution table,

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1000_1 & 1500_2 & 7000_2 \\ & 0 & 200_2 & 3000_2 \\ & & 0 & 1000_3 \\ & & & 0 \end{bmatrix} \end{array}$$

Now, how to we use the split table to determine the optimal way to group the matrices

$$A_1 \times A_2 \times A_3 \times A_4?$$

Note that the optimal split point $S[1, 4]$ is $k = 2$, which means we should split like this

$$\begin{array}{c} (A_1 \times A_2) \quad (A_3 \times A_4) \\ \uparrow \\ k = 2 \end{array}$$

For this small example there is no need to pursue the split points any farther, and we see that the optimal grouping is

$$(A_1 \times A_2) \quad (A_3 \times A_4).$$