MCS-375: Algorithms: Analysis and Design Handout #DP7
San Skulrattanakulchai
Gustavus Adolphus College Oct 19, 2015

Matrix-chain Multiplication

GT: Ch 12.1

Background

If A is a matrix of dimension $p \times q$ and B is a matrix of dimension $q \times r$, then the product AB is well-defined. It is, in fact, a matrix C of dimension $p \times r$ such that entry $c_{ij} = \sum_{k=1}^{q} a_{ik} b_{kj}$ for all $1 \leq i \leq p$ and $1 \leq j \leq r$.

The naive algorithm for multiplying two matrices using the above definition requires pqr scalar multiplications.

Matrix multiplication is associative. Given a sequence of matrices any two consecutive ones of which are compatible for multiplication, we may compute the product of the whole sequence of matrices by repeatedly replacing any two consecutive matrices by their product, until only one matrix remains.

We can specify this sequence of choices of multiplication by adding to the original sequence of matrices a pair of parentheses, for each choice of multiplication.

For example, one possible way to compute the product $A_1A_2A_3A_4A_5$ is as follows.

Step 1 Multiply A_1 by A_2.

Step 2 Multiply A_3 by A_4.

Step 3 Multiply the result matrix from Step 2 by A_5.

Step 4 Multiply the result matrix from Step 1 by the result matrix from Step 3.

This corresponds to the *fully parenthesized* sequence $((A_1A_2)((A_3A_4)A_5))$.

Note that one full parenthesization may correspond to more than one sequence of choices.

It's clear that computing the product $A_1A_2 \cdots A_n$ using different full parenthesizations may require different numbers of scalar multiplications.

Problem

Given a sequence of n matrices $A_1, A_2, \ldots, A_n$, *fully parenthesize* it to give the fewest number of scalar multiplications used in computing the product $A_1 A_2 \cdots A_n$.

Assume the dimensions of the A matrices are given in the sequence p so that, for all $1 \leq i \leq n$, the dimension of A_i is $p_{i-1} \times p_i$.

Dynamic Programming Solution

For $1 \leq i \leq j \leq n$, define $m(i, j)$ to be the fewest number of scalar multiplications required to compute the product $A_i A_{i+1} \cdots A_j$.
We seek $m(1, n)$.

Optimal Substructure Property

If $i = j$, the product $A_i A_{i+1} \cdots A_j$ is simply A_i itself and it requires no scalar multiplication to compute it. So, $m(i, j) = 0$ when $i = j$.

Suppose $i < j$. Let the very last matrix multiplication to compute the product $A_i A_{i+1} \cdots A_j$ be the multiplication of the product $A_i A_{i+1} \cdots A_k$ by the product $A_{k+1} A_{k+2} \cdots A_j$. Then the product $A_i A_{i+1} \cdots A_k$ and the product $A_{k+1} A_{k+2} \cdots A_j$ must have been computed using the fewest number of scalar multiplications as well.

Thus,

$$m(i, j) = m(i, k) + m(k + 1, j) + p_{i-1} p_k p_j.$$

We don't know what the value of k is, but we know that $i \leq k < j$.

Recurrence

The above reasoning gives the recurrence.

$$m(i, j) = \begin{cases} 0 & \text{if } i = j \\ \min\{\, m(i, k) + m(k + 1, j) + p_{i-1} p_k p_j : i \leq k < j \,\} & \text{if } i < j. \end{cases}$$

MISSING PICTURE OF $m(i, j)$ TABLE

Algorithm

Step 1. Fill in the table $m(\cdot, \cdot)$, plus a companion table of minimizers.

Step 2. Find an optimal full parenthesization by using the minimizer as a pointer to two subproblems. Since each problem has 2 subproblems, we program this recursively.

Question Suppose we fill in the $m(\cdot, \cdot)$ table row-by-row or column-by-column. Explain why the row index must be decreasing and the column index increasing.

Timing

We find an optimum parenthesization in time $O(n^3)$ because

(a) step 1 fills in each table entry in time $O(n)$, in time $O(n^3)$ total, and

(b) step 2 finds each pair of matrices to multiply in time $O(1)$, $O(n)$ time total.

This problem illustrates 2D-dynamic programming where $m(i, j)$ depends on $O(n)$ smaller values and the time is $O(n^3)$.