Dynamic Programming II - Matrix chain multiplication

1 Matrix chain multiplication problem

1.1 Matrix multiplication

Let A be a $p \times q$ matrix, and let B be a $q' \times r$ matrix, i.e.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,q} \\ a_{2,1} & a_{2,2} & \dots & a_{2,q} \\ \vdots & \vdots & \dots & \vdots \\ a_{p,1} & a_{p,2} & \dots & a_{p,q} \end{pmatrix}, \quad B = \begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,r} \\ b_{2,1} & b_{2,2} & \dots & b_{2,r} \\ \vdots & \vdots & \dots & \vdots \\ b_{q',1} & b_{q',2} & \dots & b_{q',r} \end{pmatrix}$$

The if $q = q'$ their product $C = AB$ is the $p \times r$ matrix where

$$c_{i,j} = \sum_{k=1}^{q} a_{i,k} b_{k,j}$$

If the number of columns of A is not equal to the number of rows of B, the product is not defined

1.2 Matrix multiplication algorithm

Input: a $p \times q$ matrix A ad a $q' \times r$ matrix B
Output: the product $C = AB$ if $q = q'$

Listing 1 Multiply(A,B)

```
if q ≠ q' then
    return ERROR: incompatible dimensions
else
    Let C be a new p×r matrix
    for i = 1 to p do
        for j = 1 to r do
            c_ij = 0
            for k = 1 to q do
                c_ij = c_ij = a_ik b_kj
    return C
```

Number of scalar multiplications: pqr (so n^3 if the same dimensions, as we have seen before)

1.3 Parenthesizing matrix multiplications

Matrix multiplication is associative:

$$A_1A_2A_3 = (A_1\,(A_2A_3)) = ((A_1A_2)\,A_3)$$

Let $n = 3$ and $\langle A_1, A_2, A_3 \rangle$ have the following dimensions:

$$A_1 : 10 \times 100, \quad A_2 : 100 \times 5, \quad A_3 : 5 \times 50$$

There are two ways of parenthesizing:

1. $((A_1A_2)A_3)$ requires
$$(10 \times 100 \times 5) + (10 \times 5 \times 50) = 7,500 \text{ multiplications.}$$

2. $(A_1(A_2A_3))$ requires
$$(100 \times 5 \times 50) + (10 \times 100 \times 50) = 75,000 \text{ multiplications.}$$

1.4 Matrix chain multiplication

We are given a sequence (chain) of $\langle A_1, ..., A_n \rangle$ of n compatible matrices to be multiplied: we wish to compute

$$A_1 A_2 ... A_n$$

A_i is a $p_{i-1} \times p_i$ matrix

Problem Where should we place the parentheses so as to minimise the number of operations?

Note: We are not actually computing the product here!

1.5 How many ways to parenthesize?

Let $P(n)$ denote the number of ways to parenthesize a product of n matrices for $n \geqslant 2$

$$P(2) = 1, \qquad P(3) = 2, \qquad P(4) = 5, ...$$

We have $P(n) = C_{n-1}$, where C_n is the n-th Catalan number:

$$C_n = \frac{1}{n+1}\binom{2n}{n} \sim \frac{4^n}{\sqrt{\pi} n^{3/2}}$$

Hence brute force has an exponential running time

2 Applying dynamic programming

To use dynamic programming, we follow a four step sequence:

1. Characterise the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution
4. Construct an optimal solution from the computed information

2.1 Step 1: The structure of an optimal parenthesization

Denote $A_{i..j} = A_i A_{i+1} \cdots A_j$ for $i \leqslant j$

To parenthesize $A_{i..j}$ we must split the product between A_k and A_{k+1} for some $i \leqslant k < j$

Optimal substructure: Suppose that to optimally parenthesize $A_{i..j}$ we split the product between A_k and A_{k+1}. Then both parenthesizations of $A_{1..k}$ and $A_{k+1...j}$ must be optimal

2.2 Step 2: A recursive solution

Let $m[i, j]$ be the minimum number of multiplications required to compute $A[i..j]$

Our goal is to compute $m[1, n]$

We have

$$m[i, j] = \min\left\{m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j : i \leq k < j\right\} \quad \text{for all } i < j$$

and of course

$$m[i, i] = 0 \quad \text{for all } i$$

2.3 Step 3: Computing the optimal costs

Let us use the bottom-up approach

Input: a sequence $p = \langle p_0, p_1, ..., p_n \rangle$

Output: The minimum number of multiplications required to compute the matrix chain multiplication $A_1 \cdots A_n$ where A_i has the dimension $p_{i-1} \times p_i$

Idea: Computing $m[i,j]$ for a product of $j-i+1$ matrices only uses values for products of fewer than $j-1+1$ matrices

Listing 2 MATRIX-CHAIN-ORDER(p)

```
Let m[1..n,1..n] be a new table
for i = 1 to n do
    m[i,i] = 0
for I = 2 to n do
    for i = 1 to n_I + 1 do
        j = i + I − 1
        m[i.j] = ∞
        for k = i to j − 1 do
         m[i,j] = min{m[i,k] + m[k+1,j] + p_{i−1} p_k p_j : i ≤ k < j}
return m
```

We can determine the minimum number of multiplications, but we also want to know HOW to parenthesize!

Good news: easy to modify MATRIX-CHAIN-ORDER to keep track of where we place the parentheses

We use an additional array $s[i.j]$ which keeps track of that k where we cut the product $A_{i..j}$ into $A_{i..k}$ and $A_{k_1...j}$

Listing 3 MATRIX-CHAIN-ORDER'(p)

```
Let m[1..n,1..n] and s[1..n-1, 2..n ]be new tables
for i ← 1 to n do
    m[i,i] ← 0
for I ← 2 to n do
    for i ← 1 to n_I + 1 do
        j ← i + I − 1
        m[i.j] ← ∞
        for k = i to j − 1 do
            q ← m[i,k] + m[k+1,j] + p_{i−1} p_k p_j
            if q < m[i,j] then
                m[i,j] ← q
                s[i,j] ← k
return m and s
```

2.3.1 Running time

Three nested loops: worst case running time of $O(n^3)$

By using the formula

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

one can show that the bound is tight, that is, that the worst-case running time is $\Theta(n^3)$

2.4 Step 4: Constructing an optimal solution

The table s[1..n-1,2..n] gives us the information to print out the right parenthesization

Listing 4 PRINT-PAR(s,i,j)

```
if i==j then
    print "A"_i
else
    print "("
    PRINT-PAR(s,i,s[i,j])
    PRINT-PAR(s,s[i,j+1],j)
    print ")"
```

Initial call: `PRINT-PAR(s,1,n)`