# CSE408
# Matrix Chain Multiplication

Lecture # 24

# Matrix-chain Multiplication

- Suppose we have a sequence or chain $A_1$, $A_2$, …, $A_n$ of $n$ matrices to be multiplied
  - That is, we want to compute the product $A_1A_2…A_n$
- There are many possible ways (parenthesizations) to compute the product

# Matrix-chain Multiplication …contd

- Example: consider the chain $A_1$, $A_2$, $A_3$, $A_4$ of 4 matrices
  - Let us compute the product $A_1A_2A_3A_4$
- There are 5 possible ways:
  1. $(A_1(A_2(A_3A_4)))$
  2. $(A_1((A_2A_3)A_4))$
  3. $((A_1A_2)(A_3A_4))$
  4. $((A_1(A_2A_3))A_4)$
  5. $(((A_1A_2)A_3)A_4)$

# Matrix-chain Multiplication ...contd

- To compute the number of scalar multiplications necessary, we must know:
  - Algorithm to multiply two matrices
  - Matrix dimensions

- Can you write the algorithm to multiply two matrices?

# Algorithm to Multiply 2 Matrices

**Input**: Matrices $A_{p \times q}$ and $B_{q \times r}$ (with dimensions $p \times q$ and $q \times r$)

**Result**: Matrix $C_{p \times r}$ resulting from the product $A \cdot B$

**MATRIX-MULTIPLY**$(A_{p \times q}, B_{q \times r})$

1. **for** $i \leftarrow 1$ **to** $p$
2.     **for** $j \leftarrow 1$ **to** $r$
3.         $C[i, j] \leftarrow 0$
4.         **for** $k \leftarrow 1$ **to** $q$
5.             $C[i, j] \leftarrow C[i, j] + A[i, k] \cdot B[k, j]$
6. **return** $C$

Scalar multiplication in line 5 dominates time to compute
CNumber of scalar multiplications = *pqr*

# Matrix-chain Multiplication ...contd

- Example: Consider three matrices $A_{10\times100}$, $B_{100\times5}$, and $C_{5\times50}$
- There are 2 ways to parenthesize
  - $((AB)C) = D_{10\times5} \cdot C_{5\times50}$
    - $AB \Rightarrow 10\cdot100\cdot5=5{,}000$ scalar multiplications
    - $DC \Rightarrow 10\cdot5\cdot50 =2{,}500$ scalar multiplications

Total: 7,500

  - $(A(BC)) = A_{10\times100} \cdot E_{100\times50}$
    - $BC \Rightarrow 100\cdot5\cdot50=25{,}000$ scalar multiplications
    - $AE \Rightarrow 10\cdot100\cdot50 =50{,}000$ scalar multiplications

Total: 75,000

# Matrix-chain Multiplication …contd

- Matrix-chain multiplication problem
  - Given a chain $A_1, A_2, \ldots, A_n$ of $n$ matrices, where for $i$=1, 2, …, $n$, matrix $A_i$ has dimension $p_{i-1} \times p_i$
  - Parenthesize the product $A_1A_2 \ldots A_n$ such that the total number of scalar multiplications is minimized
- Brute force method of exhaustive search takes time exponential in $n$

# Dynamic Programming Approach

- The structure of an optimal solution
  - Let us use the notation $A_{i..j}$ for the matrix that results from the product $A_i A_{i+1} \dots A_j$
  - An optimal parenthesization of the product $A_1A_2 \dots A_n$ splits the product between $A_k$ and $A_{k+1}$ for some integer $k$ where $1 \le k < n$
  - First compute matrices $A_{1..k}$ and $A_{k+1..n}$ ; then multiply them to get the final matrix $A_{1..n}$

# Dynamic Programming Approach …contd

- **Key observation**: parenthesizations of the subchains $A_1A_2 \ldots A_k$ and $A_{k+1}A_{k+2} \ldots A_n$ must also be optimal if the parenthesization of the chain $A_1A_2 \ldots A_n$ is optimal (why?)

- That is, the optimal solution to the problem contains within it the optimal solution to subproblems

# Dynamic Programming Approach …contd

- Recursive definition of the value of an optimal solution
  - Let $m[i, j]$ be the minimum number of scalar multiplications necessary to compute $A_{i..j}$
  - Minimum cost to compute $A_{1..n}$ is $m[1, n]$
  - Suppose the optimal parenthesization of $A_{i..j}$ splits the product between $A_k$ and $A_{k+1}$ for some integer $k$ where $i \leq k < j$

# Dynamic Programming Approach ...contd

- $A_{i..j} = (A_i A_{i+1} \ldots A_k) \cdot (A_{k+1} A_{k+2} \ldots A_j) = A_{i..k} \cdot A_{k+1..j}$
- Cost of computing $A_{i..j}$ = cost of computing $A_{i..k}$ + cost of computing $A_{k+1..j}$ + cost of multiplying $A_{i..k}$ and $A_{k+1..j}$
- Cost of multiplying $A_{i..k}$ and $A_{k+1..j}$ is $p_{i-1} p_k p_j$

- $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$ *for* $i \le k < j$
- $m[i, i] = 0$ for $i=1,2,\ldots,n$

# Dynamic Programming Approach …contd

– But… optimal parenthesization occurs at one value of k among all possible $i \le k < j$
– Check all these and select the best one

$$m[i, j] = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1} p_k p_j\} & \text{if } i<j \end{cases}$$

# Dynamic Programming Approach ...contd

- To keep track of how to construct an optimal solution, we use a table *s*
- $s[i, j]$ = value of $k$ at which $A_i A_{i+1} \ldots A_j$ is split for optimal parenthesization
- Algorithm: next slide
  - First computes costs for chains of length $l$=1
  - Then for chains of length $l$=2,3, … and so on
  - Computes the optimal cost bottom-up

# Algorithm to Compute Optimal Cost

**Input**: Array $p[0\ldots n]$ containing matrix dimensions and $n$
**Result**: Minimum-cost table $m$ and split table $s$

**MATRIX-CHAIN-ORDER**($p[\ ]$, $n$)

```
for i ← 1 to n
    m[i, i] ← 0
for l ← 2 to n
    for i ← 1 to n-l+1
        j ← i+l-1
        m[i, j] ← ∞
        for k ← i to j-1
            q ← m[i, k] + m[k+1, j] + p[i-1] p[k] p[j]
            if q < m[i, j]
                m[i, j] ← q
                s[i, j] ← k
return m and s
```

Takes $O(n^3)$ time

Requires $O(n^2)$ space

# Constructing Optimal Solution

- Our algorithm computes the minimum-cost table *m* and the split table *s*
- The optimal solution can be constructed from the split table *s*
  - Each entry $s[i, j] = k$ shows where to split the product $A_i A_{i+1} \ldots A_j$ for the minimum cost

# Example

- Show how to multiply this matrix chain optimally
- Solution on the board
  - Minimum cost 15,125
  - Optimal parenthesization $((A_1(A_2A_3))((A_4A_5)A_6))$

| Matrix | Dimension |
|---|---|
| $A_1$ | 30×35 |
| $A_2$ | 35×15 |
| $A_3$ | 15×5 |
| $A_4$ | 5×10 |
| $A_5$ | 10×20 |
| $A_6$ | 20×25 |

Thank You !!!