DYNAMIC PROGRAMMING: MATRIX CHAIN MULTIPLICATION

Instructor: Dr Tarunpreet Bhatia
Assistant Professor, CSED
TIET

Matrix-chain Multiplication

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

Matrix-chain Multiplication

- 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

- 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 C
Number of scalar multiplications = pqr

Matrix Multiplication of ABC

Given a $p \times q$ matrix A and a $q \times r$ matrix B and a $r \times s$ matrix C, then ABC can be computed in two ways $(AB)C$ and $\boldsymbol{A(BC)}$:

The number of multiplications needed are:

$$mult[\,(AB)C\,] = pqr + prs$$
$$mult[\,A(BC)\,] = qrs + pqs$$

When $p = 5$, $q = 4$, $r = 6$ and $s = 2$, then

$$mult[\,(AB)C\,] = 180$$
$$mult[\,A(BC)\,] = 88$$

A big difference!

Implication: The multiplication “sequence” (parenthesis) is important

Matrix-chain Multiplication …contd

- Matrix-chain multiplication problem
  – Given a chain $A_1, A_2, \ldots, A_n$ of n matrices, where for $i=1, 2, \ldots, n$, 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

- Step 1: The structure of an optimal solution
  - Let us use the notation $A_{i..j}$ for the matrix that results from the product $A_i A_{i+1} \ldots A_j$
  - An optimal parenthesization of the product $A_1A_2\ldots A_n$ 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 $A_{1..n}$

Step 1: The structure of an optimal solution

Split the original structure into substructures

Suppose $A_i, A_{i+1}, \ldots, A_j$ is split between two substructures around k

$A_i, A_{i+1}, \ldots, A_j$

Split into 2 substructures

$\boldsymbol{A_i, A_{i+1}, \ldots, A_k}$ $\boldsymbol{A_{k+1}, \ldots, A_j}$

- Find the optimal solution for both substructures
- And then combine them to get the optimal solution of the original structure

NOTE: Need to ensure the correct place (*i.e.*, *k*) to split the product

Dynamic Programming Approach

- **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.

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

Dynamic Programming Approach

- Step 2: 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 $i \le k < j$.
  - $A_{i..j} = (A_i A_{i+1} \ldots A_k) \cdot (A_{k+1} A_{k+2} \ldots A_j) = A_{i..k} \cdot A_{k+1..j}$
  - 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 $p_{i-1} p_k p_j$

Dynamic Programming Approach

- Optimal parenthesization occurs at one value of k among all possible $i \leq k < j$
- Check all these and select the best one

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

Dynamic Programming Approach

- To keep track of how to construct an optimal solution, we use a table *s*
- $s[i, j]$ = value of k at which $A_i A_{i+1} \dots A_j$ 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

Step 3: 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

Example 1

```
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
```

Example 1

```
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
```

Step 4: 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

Constructing Optimal Solution

- Algorithm to Construct an optimal solution from computed information

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

Constructing Optimal Solution

```
Print-Optimal-Parens(s,1,3)
1. if 1= 3
2.     print A1
3. else Print "("
4.     print-Optimal-Parens(s,1,1)
5.     print-Optimal-Parens(s,2,3)
6.     print ")"
```

```
Print-Optimal-Parens(s,1,1)
1. if 1= 1
2.     print A1
3. else ..
4.     ..
5.     ..
6.     ..
```

```
Print-Optimal-Parens(s,2,2)
1. if 2 = 2
2.     print A2
3. else ..
4.     ..
5.     ..
6.     ..
```

```
Print-Optimal-Parens(s,2,3)
1. if 2= 3
2.   print A2
3. else Print "("
4.     Print-Optimal-Parens(s,2,2)
5.     Print-Optimal-Parens(s,3,3)
6.     print ")"
```

```
Print-Optimal-Parens(s,3,3)
1. if 3 = 3
2.     print A3
3. else ..
4.     ..
5.     ..
6.     ..
```

Constructing Optimal Solution

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

Try this!

Given a sequence of 4 matrices M_i ($1 \le i \le 4$) and p[ ]= {5, 4, 6, 2, 7}. What will be final order of parentheses so that the product of the matrices, in order, is unambiguous and needs the minimal number of multiplications.

$p_0 = 5, p_1 = 4, p_2 = 6, p_3 = 2$ and $p_4 = 7$

Solution

Total Multiplication = $\boldsymbol{m}[\mathbf{1},\mathbf{4}]$ = 158

$((\boldsymbol{A1})\ (\boldsymbol{A2}\ \boldsymbol{A3}))\ (\boldsymbol{A4})$

		1	2	3	4
	4	158	104	84	0
	3	88	48	0	
i	2	120	0		
	1	0			

		1	2	3
	4	3	3	3
j	3	1	2	
	2	1		

Example 2

Given a sequence of 5 matrices M_i ($1 \le i \le 5$) and p[ ] = {4,10,3,12,20,7}. What will be final order of parentheses so that the product of the matrices, in order, is unambiguous and needs the minimal number of multiplications.

Example 3

- Consider the following six matrix problem.

Matrix	A_1	A_2	A_3	A_4	A_5	A_6
Dimensions	10x20	20x5	5x15	15x50	50x10	10x15

- The problem therefore can be phrased as one of filling in the following table representing the values m.

i\j	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Example 3 (Contd.)

- Chains of length 2 are easy, as there is no minimization required, so
- $m[i, i+1] = p_{i-1}p_ip_{i+1}$
- m[1, 2] = 10x20x5 = 1000
- m[2, 3] = 20x5x15 = 1500
- m[3, 4] = 5x15x50 = 3750
- m[4, 5] = 15x50x10 = 7500
- m[5, 6] = 50x10x15 = 7500

i\j	1	2	3	4	5	6
1	0	1000				
2		0	1500			
3			0	3750		
4				0	7500	
5					0	7500
6						0

Example 3 (contd.)

- Chains of length 3 require some minimization – but only one each.
- $m[1,3]=\min\{(m[1,1]+m[2,3]+p_0p_1p_3),(m[1,2]+m[3,3]+p_0p_2p_3)\}$
  $= \min\{(0+1500+10x20x15), (1000+0+10x5x15)\}$
  $= \min \{ 4500, 1750 \} = 1750$
- $m[2,4]=\min\{(m[2,2]+m[3,4]+p_1p_2p_4),(m[2,3]+m[4,4]+p_1p_3p_4)\}$
  $= \min\{(0+3750+20x5x50), (1500+0+20x15x50)\}$
  $= \min \{ 8750, 16500 \} = 8750$
- $m[3,5]=\min\{(m[3,3]+m[4,5]+p_2p_3p_5),(m[3,4]+m[5,5]+p_2p_4p_5)\}$
  $= \min\{(0+7500+5x15x10), (3750+0+5x50x10)\}$
  $= \min \{ 8250, 6250 \} = 6250$
- $m[4,6]=\min\{(m[4,4]+m[5,6]+p_3p_4p_6),(m[4,5]+m[6,6]+p_3p_5p_6)\}$
  $= \min\{(0+7500+15x50x15), (7500+0+15x10x15)\}$
  $= \min \{ 18750, 9750 \} = 9750$

i\j	1	2	3	4	5	6
1	0	1000	1750			
2		0	1500	8750		
3			0	3750	6250	
4				0	7500	9750
5					0	7500
6						0

Example 3 (contd.)

- $m[1,4]=\min\{(m[1,1]+m[2,4]+p_0p_1p_4),(m[1,2]+m[3,4]+p_0p_2p_4),$
  $(m[1,3]+m[4,4]+p_0p_3p_4)\}$
  $= \min\{(0+8750+10x20x50), (1000+3750+10x5x50),$
  $(1750+0+10x15x50)\}$
  $= \min \{ 18750, 7250, 9250 \} = 7250$
- $m[2,5]=\min\{(m[2,2]+m[3,5]+p_1p_2p_5),(m[2,3]+m[4,5]+p_1p_3p_5),$
  $(m[2,4]+m[5,5]+p_1p_4p_5)\}$
  $= \min\{(0+6250+20x5x10), (1500+7500+20x15x10),$
  $(8750+0+20x50x10)\}$
  $= \min \{ 7250, 12000, 18750 \} = 7250$
- $m[3,6]=\min\{(m[3,3]+m[4,6]+p_2p_3p_6),(m[3,4]+m[5,6]+p_2p_4p_6),$
  $(m[3,5]+m[6,6]+p_2p_5p_6)\}$
  $= \min\{(0+9750+5x15x15), (3750+7500+5x50x15),$
  $(6250+0+5x10x15)\}$
  $= \min \{ 10875, 15000, 7000 \} = 7000$

i\j	1	2	3	4	5	6
1	0	1000	1750	7250		
2		0	1500	8750	7250	
3			0	3750	6250	7000
4				0	7500	9750
5					0	7500
6						0

Example 3 (contd.)

- $m[1,5]=\min\{(m[1,1]+m[2,5]+p_0p_1p_5),(m[1,2]+m[3,5]+p_0p_2p_5),$
  $(m[1,3]+m[4,5]+p_0p_3p_5),(m[1,4]+m[5,5]+p_0p_4p_5)\}$
  $= \min\{(0+7250+10x20x10), (1000+6250+10x5x10),$
  $(1750+7500+10x15x10), (7250+0+10x50x10)\}$
  $= \min \{ 9250, 7750, 10750, 12250 \} = 7750$
- $m[2,6]=\min\{(m[2,2]+m[3,6]+p_1p_2p_6),(m[2,3]+m[4,6]+p_1p_3p_6),$
  $(m[2,4]+m[5,6]+p_1p_4p_6),(m[2,5]+m[6,6]+p_1p_5p_6)\}$
  $= \min\{(0+7000+20x5x15), (1500+9750+20x15x15),$
  $(8750+7500+20x50x15), (7250+0+20x10x15)\}$
  $= \min \{ 8500, 15750, 31,250, 10250 \} = 8500$
- $m[1,6]=\min\{(m[1,1]+m[2,6]+p_0p_1p_6),(m[1,2]+m[3,6]+p_0p_2p_6),$
  $(m[1,3]+m[4,6]+p_0p_3p_6),(m[1,4]+m[5,6]+p_0p_4p_6),$
  $(m[1,5]+m[6,6]+p_0p_5p_6)\}$
  $= \min\{(11500, 8750, 13750, 22250, 9250 \} = 8750$

i\j	1	2	3	4	5	6
1	0	1000	1750	7250	7750	8750
2		0	1500	8750	7250	8500
3			0	3750	6250	7000
4				0	7500	9750
5					0	7500
6						0

Example 3 (contd.)

- So far we have decided that the best way to parenthesize the expression results in 8750 multiplication.
- But we have not addressed how we should actually DO the multiplication to achieve the value.
- However, look at the last computation we did – the minimum value came from computing

$$A = (A_1A_2)(A_3A_4A_5A_6)$$

- Therefore in an auxiliary array, we store value s[1,6]=2.
- In general, as we proceed with the algorithm, if we find that the best way to compute $A_{i..j}$ is as

$$A_{i..j} = A_{i..k}A_{(k+1)..j}$$

  then we set

$$s[i, j] = k.$$

- Then from the values of k we can reconstruct the optimal way to parenthesize the expression.

Example 3 (contd.)

- If we do this then we find that the s array looks like this:

i\j	1	2	3	4	5	6
1	1	1	2	2	2	2
2		2	2	2	2	2
3			3	3	4	5
4				4	4	5
5					5	5
6						6

- We already know that we must compute $A_{1..2}$ and $A_{3..6}$.
- By looking at s[3,6] = 5, we discover that we should compute $A_{3..6}$ as $A_{3..5}A_{6..6}$ and then by seeing that s[3,5] = 4, we get the final parenthesization

$$A = ((A_1A_2)(((A_3A_4)A_5)A_6)).$$

- And quick check reveals that this indeed takes the required 8750 multiplications.

Example 3 (contd.)

i\j	1	2	3	4	5	6
1	1	1	2	2	2	2
2		2	2	2	2	2
3			3	3	4	5
4				4	4	5
5					5	5
6						6