Dynamic Programming

Matrix Chain Multiplication

Dynamic Programming

- Solve Optimization Problem
- Bottom-up Approach
  - Start at the bottom
  - Solve small subproblems
  - Store solutions to table/array
  - Reuse previous results of subproblems for solving larger subproblems

Matrix-chain multiplication

- We are given a sequence (chain)

  $<A_1, A_2, \ldots, A_n>$

  to compute the product

  $A_1 A_2 \ldots A_n$
- Matrix multiplication is **associative**, and so all parenthesizations yield the same product.

- A product of matrices is ***fully parenthesized***
  - if it is either a single matrix or the product of two fullyparenthesized matrix products, surrounded by parentheses.

Matrix-chain multiplication(cont.)

- For example, if the chain of matrices is $\langle A_1, A_2, A_3, A_4 \rangle$, then we can fully parenthesize the product $A_1A_2A_3A_4$ in five distinct ways:

$(A_1 (A_2(A_3 A_4)))$
$(A_1 ((A_2 A_3)A_4))$
$((A_1A_2)(A_3 A_4))$
$((A_1 (A_2A_3))A_4)$
$(((A_1 A_2)A_3) A_4)$

Matrix-chain multiplication

```
Matrix_Multiply(Apxq,Bqxr)
  for i=1 to p
    for j=1 to r
            Cij=0
            for k=1 to q
                Cij=Cij+Aik.Bkj
   return C
```

Example

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \\ 2 & 3 & 4 \\ 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 1 & 1 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 7 & 13 \\ 4 & 6 \\ 13 & 19 \\ 4 & 6 \end{pmatrix}$$

4X3 3X2 4X2

Total Multiplications = 4x3x2=24

Matrix-chain multiplication

- The time to compute C is dominate by the number of scalar multiplications, which is pqr.

- Express **costs in terms of the number of scalar multiplications**.

- The costs obtained by different parenthesizations of a matrix product

Matrix-chain multiplication(cont.)

- consider the problem of a chain < A_1, A_2, A_3>of three matrices.
  - dimensions of the matrices
  - A_1: 10 X 100 A_2: 100 X 5, A_3: 5 X 50.
  - Multiply $((A_1A_2)A_3)$,
    - A_1A_2= 10 X 100 X5 = 5000 scalar multiplications
    - $(A_1A_2)A_3$= 10 X 5 X 50 = 2500 scalar multiplications
    - Total of 7500 scalar multiplications.
  - Multiply $(A_1 (A_2A_3))$,
    - A_2A_3= 100 X 5 X 50 =25,000 scalar multiplications
    - $A_1 (A_2A_3)$= 10 X 100 X 50 = 50,000 scalar multiplications
    - Total of 75,000 scalar multiplications.
  - Thus, computing the product according to the first parenthesization is 10 times faster.

Definition: 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} X p_i , fully parenthesize the product $A_1A_2 \ldots A_n$ in a way that minimizes the number of scalar multiplications.
- Our goal is **to determine an order for multiplying matrices that has the lowest cost.**

Developing a dynamic programming Algorithm

- **Steps of Dynamic programming**

To **find the optimal substructure** and then use it to construct an optimal solution to the problem **from optimal solutions to subproblems.**

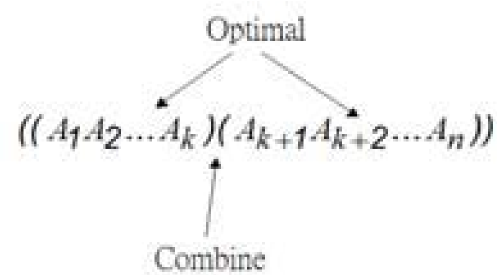
1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution
4. Construct an optimal solution from computed information.

Step 1: The structure of an optimal parenthesization

- In the matrix-chain multiplication problem, we can perform this step as follows.
  - **Decompose the problem into subproblems :** For each pair $1 \le i \le j \le n$, determine the multiplication sequence for $A_{i..j} = A_i A_{i+1} \ldots A_j$ that minimizes the number of multiplications, $A_{i..j}$ is $p_{i-1} \times p_j$ matrix

The optimal substructure Property:

- Suppose that to optimally parenthesize $A_iA_{i+1}.....A_j$, **we split the product between A_k and A_{k+1}, for some integer k in the range $i \leq k < j$.**
- Then parenthesize the "prefix" subchain $A_iA_{i+1}.....A_k$ within this optimal parenthesization of $A_iA_{i+1}.....A_j$, it must be an optimal parenthesization of $A_iA_{i+1}.....A_k$.
- we parenthesize the subchain $A_{k+1} A_{k+2}.....A_j$ in the optimal parenthesization of $A_iA_{i+1}.....A_j$, it must be an optimal parenthesization of $A_{k+1} A_{k+2}.....A_j$.



Step 2: A recursive solution

- Define the cost of an optimal solution recursively in terms of the optimal $A_i A_{i+1}.....A_j$ solutions to subproblems.
- Let m[i,j] be the minimum number of scalar multiplication needed to compute the matrix $A_i A_{i+1}.....A_j$ for $1 \le i \le j \le n$.
- *split the product between* $\boldsymbol{A_k}$ ***and*** $\boldsymbol{A_{k+1}}$, *for some integer* ***k*** *in the range* $\boldsymbol{i \le k < j}$.
- *m[i,j]= The min. cost for computing* $A_{i..k}$ *+ The min. cost for computing* $A_{k+1..j}$ *+ the cost of multiplying* $A_{i..k}$ *and* $A_{k+1..j}$ *matrices*.

- Recursive definition for the minimum cost of parenthesizing the product $A_iA_{i+1}.....A_j$ is

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

The $m[i, j]$ values give the costs of optimal solutions to subproblems.

- To construct an optimal solution, we define s[i, j] to be a value of k at which we split the product $A_iA_{i+1}.....A_j$ in an optimal parenthesization.

Step 3: Computing the optimal costs

- Compute the optimal cost by using a tabular, bottom-up approach
- MATRIX-CHAIN- ORDER procedure assumes that matrix A_i has dimensions p_{i-1} X p_i for i=1,2,..,n.
  - The procedure uses an auxiliary table m[1.. n, 1.. n] for storing the m[i, j] costs
  - Auxiliary table S[1.. n- 1, 2 .. n] that records which index of k achieved the optimal cost in computing m[i..j].
  - m[i, j] = the minimum cost for computing the subproducts $A_{i..k}$ and $A_{k+1..j}$ + the cost of multiplying these two matrices together.
  - Use the table s to construct an optimal solution.
  - In order to implement the bottom-up approach, we must determine which entries of the table we refer to when computing m[i..j].

MATRIX-CHAIN-ORDER(p)

```
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i,i] = 0
for l = 2 to n              // l is the chain length
    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
```

MATRIX-CHAIN-ORDER- time complexity- $O(n^3)$

Space complexity- $\Theta(n^2)$ - to store the m and s tables.

Step 4: Constructing an optimal solution

- An optimal solution can be constructed from the computed information stored in the table
- The final matrix multiplication in computing $A_{1..n}$ optimally is $A_{1..s[1,n]}A_{s[1,n]+1..n.}$

- The following recursive procedure prints an optimal parenthesization of $A_iA_{i+1}.....A_j$, given the s table computed by MATRIX-CHAIN-ORDER and the indices i and j.

- The initial call PRINT-OPTIMAL- PARENS(s,1,n) prints an optimal parenthesization of $<A_1,A_2,....,A_n>$.

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

- The call PRINT-OPTIMAL-PARENS(s, 1, 6) prints the parenthesization

$$((A_1(A_2A_3))((A_4A_5)A_6)).$$

Example

- Given a chain of 4 matrices <A_1,A_2,A_3,A_4> with dimensions <5X4>,<4X6>,<6X2>,<2X7> respectively. Using Dynamic programming find the minimum number of scalar multiplications needed and also write the optimal multiplication order. (December 2019-5 marks)
- A_1 A_2 A_3 A_4
- P_0 P_1 P_2 P_3 P_4
- 5 4 6 2 7

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

The $m[i, j]$ values give the costs of optimal solutions to subproblems

- Step 1:Fill the table for i=j
  - M[1,1]=0
  - M[2,2]=0
  - M[3,3]=0
  - M[4,4]=0

m

i \ j	1	2	3	4
1	0			
2	x	0		
3	x	x	0	
4	x	x	x	0

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

The $m[i,j]$ values give the costs of optimal solutions to subproblems

S

i/j	2	3	4
1	1		
2	x	2	
3	x	x	3

m

i \ j	1	2	3	4
1	0	120		
2	x	0	48	
3	x	x	0	84
4	x	x	x	0

- Step 1:Fill the table for
  - i=1,j=2 ⇨ $M[1,2]=\min_{1\le k<2}\ \{m[1,1]+m[2,2]+P_0 P_1 P_2\}$ =0+0+5x4x6=120
  - i=2,j=3 ⇨ $M[2,3]= \min_{2\le k<3}\ \{m[2,2]+m[3,3]+P_1 P_2 P_3\}$ =0+0+4x6x2=48
  - i=3,j=4 ⇨ $M[3,4]= \min_{3\le k<4}\ \{m[3,3]+m[4,4]+P_2 P_3 P_4\}$ =0+0+6x2x7=84

- Step 1:Fill the table for
  - $i=1,j=3$
  - $M[1,3]=\min_{1\le k<3}$ {
    - K=1,

m

i \ j	1	2	3	4
1	0	120	88	
2	x	0	48	
3	x	x	0	84
4	x	x	x	0

$m[1,3]=m[1,1]+m[2,3]+ P_0 P_1 P3 =0+48+5x4x2=88$

- k=2,

$M[1,3]= m[1,2]+m[3,3]+ P_0 P_2 P_3\} =120+0+5x6x2=180$

- Therefore m[1,3]=min{88,180}=88

s

i/j	2	3	4
1	1	1	
2	x	2	
3	x	x	3

- Step 1:Fill the table for
  - i=2,j=4
  - $M[2,4]=\min_{2 \le k < 4}$ {
    - K=2,

      $m[2,4]=m[2,2]+m[3,4]+P_1 P_2 P_4 = 0+84+4x6x7=252$
    - K=3,

      $M[2,4]= m[2,3]+m[4,4]+P_1P_3P_4\} =48+0+4x2x7=104$
  - Therefore m[2,4]=min{252,104}=104

m

i \ j	1	2	3	4
1	0	120	88	
2	x	0	48	104
3	x	x	0	84
4	x	x	x	0

S

i/j	2	3	4
1	1	1	
2	x	2	3
3	x	x	3

m

i \ j	1	2	3	4
1	0	120	88	158
2	x	0	48	104
3	x	x	0	84
4	x	x	X	0

- Step 1:Fill the table for
  - i=1,j=4
  - $M[1,4]=\min_{1\leq k<4}$ {
    - K=1

      $m[1,4]=m[1,1]+m[2,4]+P_0P_1P_4 = 0+104+5x4x7=244$
    - K=2,

      $m[1,4]=m[1,2]+m[3,4]+P_0P_2P_4 = 120+84+5x6x7=414$
    - K=3,

      $M[1,4]= m[1,3]+m[4,4]+P_0P_3P_4\} = 88+0+5x2x7=158$
  - Therefore m[2,4]=min{244,414,158}

    =158

S

i/j	2	3	4
1	1	1	3
2	x	2	3
3	x	x	3

– Parenthesization of

$A_1 \quad A_2 \qquad A_3 \qquad\quad A_4$

By calling

m

i \ j	1	2	3	4
1	0	120	88	158
2	x	0	48	104
3	x	x	0	84
4	x	x	X	0

PRINT-OPTIMAL-PARENS(s, 1, 4) we will get

$M[1,4]= m[1,3]+m[4,4]+ P_0P_3P_4\}$

$((A_1 \quad A_2 \qquad A_3 \ ) A_4 \ )$

$m[1,3]=m[1,1]+m[2,3]+ P_0 P_1P3$

$((A_1 \quad (A_2 \qquad A_3 \ )) A_4 \ )$

- Optimal scalar multiplications -158
- Order-((A1(A2A3))A4)

S

i/j	2	3	4
1	1	1	3
2	x	2	3
3	x	x	3

2)Using Dynamic Programming, find the fully parenthesized matrix product for multiplying the chain of matrices< A1 A2 A3 A4 A5 A6 > whose dimensions are <30X35>, <35X15>,<15X5>, <5X10>, <10X20> and <20X25> respectively. (May 2019-5 marks, April 2018-5 marks)

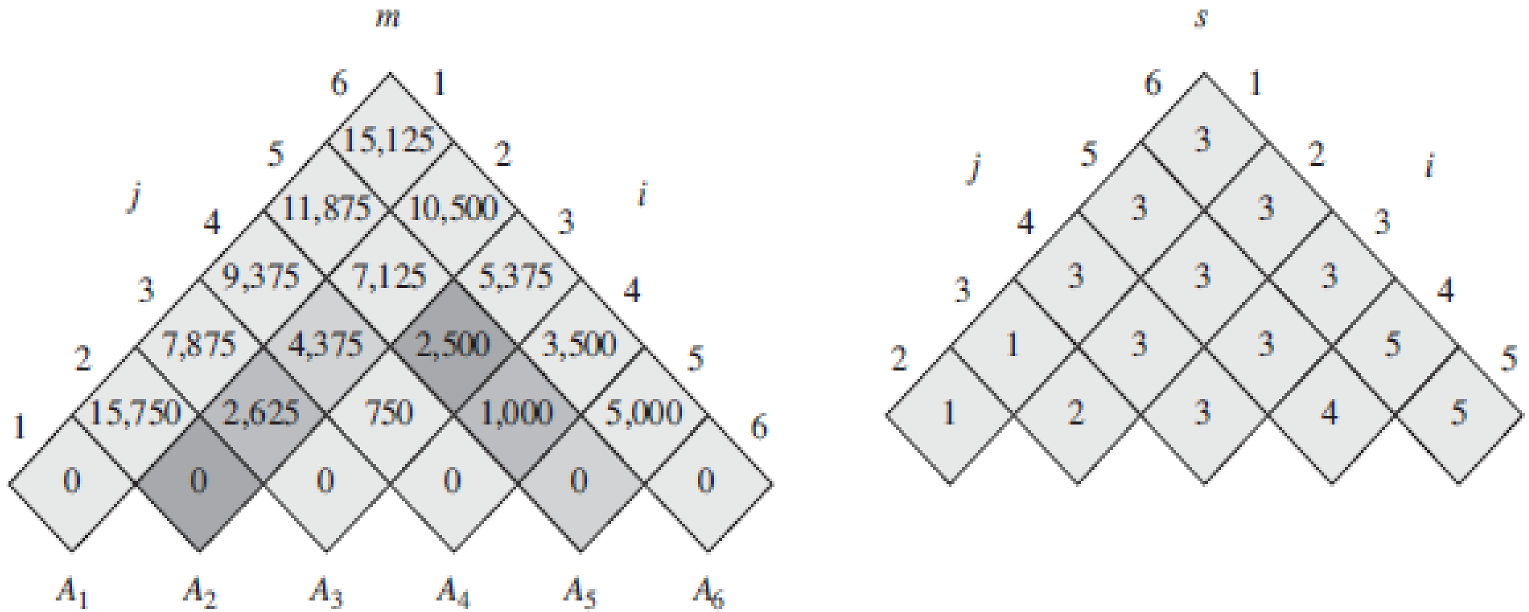


Figure . The m and s tables computed by MATRIX-CHAIN-ORDER for $n = 6$ and the following matrix dimensions:

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

The tables are rotated so that the main diagonal runs horizontally. The m table uses only the main diagonal and upper triangle, and the s table uses only the upper triangle. The minimum number of scalar multiplications to multiply the 6 matrices is $m[1,6] = 15{,}125$. Of the darker entries, the pairs that have the same shading are taken together in line 10 when computing

$$
\begin{aligned}
m[2,5] &= \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 &= 13{,}000 , \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 &= 7125 , \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 &= 11{,}375 \end{cases} \\
&= 7125 .
\end{aligned}
$$

- The call PRINT-OPTIMAL-PARENS(s, 1, 6) prints the parenthesization

$$((A_1(A_2A_3))((A_4A_5)A_6)).$$