Matrix-chain Multiplication

We are given a sequence (chain) $\langle A_1, A_2, \ldots A_n \rangle$ of 'n' matrices to be multiplied, and we wish to compute the product

$$A_1 A_2 \cdots A_n$$

We can evaluate the above expression using the standard alg for multiplying pairs of matrices as a subroutine once we have parenthesized it to resolve all ambiguities in how the matrices are multiplied together. Matrix multiplication is associative, and so all parenthesizations yields the same product.

Example :- The chain of matrices is $\langle A_1, A_2, A_3, A_4 \rangle$, the product $A_1A_2A_3A_4$ can be fully parenthesized in five diff ways.

$(A_1(A_2(A_3A_4)))$,

$(A_1((A_2A_3)A_4))$,

$((A_1A_2)(A_3A_4))$,

$((A_1(A_2A_3))A_4)$,

$(((A_1A_2)A_3)A_4)$.

*. [The way we parenthesize a chain of matrices can have a dramatic impact on the cost of evaluating the product]. Consider the cost of multiplying two matrices

```
Matrix_Multiply (A,B)
1  if cols[A] ≠ rows [B]
2  then error "incompatible dimensions"
3  else for i ← 1 to rows[A] do
4          for j ← 1 to cols[B] do
5              C[i,j] ← 0
6          for k ← 1 to cols[A] do
7              C[i,j] ← C[i,j] + A[i,k] * B[k,j];
8  return C
```

We can multiply two matrices A and B iff they are compatible : the no. of columns of A must equal to no. of rows of B. If A is a $p \times q$ matrix and B is a $q \times r$ matrix, the resulting matrix C is a $p \times r$ matrix. The time to compute C is dominated by the no. of scalar multiplications in line 7, which is pqr. Hence we shall express the cost in terms of no. of scalar multiplications.

Example :- To illustrate the diff costs incurred by diff parenthesizations of a matrix product, consider the problem of chain $\langle A_1, A_2, A_3 \rangle$ of three matrices. Suppose that the dimensions of the matrices are 10×100, 100×5 & 5×50, respectively.

If we multiply according to the parenthesization $((A_1 A_2) A_3)$, we ~~pre~~ perform $10 * 100 * 5 = 5000$ scalar multiplications to compute 10×5 matrix product $A_1 A_2$, plus another $10 * 5 * 50 = 2500$ scalar multiplications to multiply the matrix A_3 with the product $A_1 A_2$, for a total of 7500 scalar multiplications.

If we instead multiply according to the parenthesization $(A_1 (A_2 A_3))$, we perform $100 * 5 * 50 = 25{,}000$ scalar multiplications to compute 100×50 matrix product $A_2 A_3$, plus another $10 * 100 * 50 = 50{,}000$ scalar multiplications to multiply the matrix A_1 with the product $A_2 A_3$, for a total of 75,000 scalar multiplications.

Thus, computing the product according to the first parenthesization is 10 times faster.

[The matrix-chain multiplication problem can be stated as follows: given a chain $\langle A_1, A_2 \cdots A_n \rangle$ of 'n' matrices, where for $i = 1, 2, 3 \cdots n$, matrix A_i has dimension $P_{i-1} \times P_i$, fully parenthesize the product $A_1 A_2 A_3 \cdots A_n$ in a way that minimizes the number of scalar multiplications]

[Note that in the ~~the~~ matrix-chain multiplication problem, we are not actually multiplying the matrices. Our goal is only to determine an order for multiplying matrices that has the lowest cost] Typically, the time invested in determining this optimal order is more than paid for by the time saved later on when actually performing the matrix multiplications (such as performing only 7500 scalar multiplications instead of 75,000).

Scanned with CamScanner

Structure of optimal parenthesization:

(Let us adopt the notation $A_{i..j}$, where $i \le j$, for the matrix that results from evaluating the product $A_i A_{i+1} \cdots A_j$. Observe that if the problem is nontrivial, i.e $i<j$, then any parenthesization of the product $A_i A_{i+1} \cdots A_j$ must split the product between A_k and A_{k+1} for some integer k in the range $i \le k < j$. That is for some value of k, we first compute the matrices $A_{i..k}$ and $A_{k+1..j}$ and then multiply them together to produce the final product $A_{i..j}$. The cost of this parenthesization is thus the cost of computing the matrix $A_{i..k}$, plus cost of computing $A_{k+1..j}$, plus the cost of multiplying them together).

We construct an optimal solution to the problem from optimal solutions to subproblems. Any optimal solution contains within it optimal solutions to its subproblems. Thus we can build an optimal solution to an instance of the matrix-chain multiplication problem by splitting the problem into two subproblems (optimally parenthesizing $A_i A_{i+1} \cdots A_k$ and $A_{k+1} A_{k+2} \cdots A_j$), finding optimal solutions to subproblem instances, and then combining these optimal ~~solutions~~ subproblem solutions.

Suppose that an optimal parenthesization of $A_i A_{i+1} \cdots A_j$ splits the product between A_k and A_{k+1}. Then the parenthesization of the prefix subchain $A_i A_{i+1} \cdots A_k$ within this optimal parenthesization of $A_i A_{i+1} \cdots A_j$ must be an optimal parenthesization of $A_i A_{i+1} \cdots A_k$. Why? If there were a less costly way to parenthesize $A_i A_{i+1} \cdots A_k$, substituting that parenthesization in the optimal parenthesization of $A_i A_{i+1} \cdots A_j$ would produce another parenthesization of $A_i A_{i+1} \cdots A_j$. whose cost was lower than the optimum. A similar observation holds for the parenthesization of the subchain $A_{k+1} A_{k+2} \cdots A_j$ in the optimal parenthesization of $A_i A_{i+1} \cdots A_j$. it must be an optimal parenthesization of $A_{k+1} A_{k+2} \cdots A_j$.

A Recursive Solution:

We can define the cost of an optimal solution recursively in terms of the optimal solutions to subproblems.

[Let $m[i,j]$ be the minimum no. of scalar multiplications needed to compute the matrix $A_{i \ldots j}$] for the full problem the cost of a cheapest way to compute $A_{1 \ldots n}$ would thus be $m[1,n]$ ~~A~~]

[We can define $m[i,j]$ recursively as follows: If $i=j$, the problem is trivial; the chain consists of just one matrix $A_{i \ldots i} = A_i$, so that no scalar multiplications are needed to compute the product. Thus $m[i,i] = 0$ for $i = 1, 2, \ldots n$). To compute $m[i,j]$ when $i<j$; we take advantage of the structure of an optimal ~~product $A_i A_{i+1} \cdots A_j$~~ solution]

[Let us assume that the optimal parenthesization splits the product $A_i A_{i+1} \cdots A_j$ between A_k and A_{k+1}, where $i \le k < j$. Then, $m[i,j]$ is equal to the minimum cost for computing the subproducts $A_{i \ldots k}$ and $A_{k+1} \ldots A_j$, plus the cost of multiplying these two matrices] together. Recalling that each matrix A_i is $P_{i-1} \times P_i$, we see that computing the matrix product $A_{i..k} A_{k+1..j}$ takes $P_{i-1} P_k P_j$ scalar multiplications. Thus we obtain

$$P_{i-1} \times P_k \quad * \quad P_k \times P_j \quad = \quad P_{i-1} P_k P_j$$

$$m[i,j] = m[i,k] + m[k+1,j] + P_{i-1} P_k P_j$$

There are only $j-i$ possible values for k, however namely $k = i, i+1 \ldots, j-1$. Since the optimal parenthesization must use one of these values for k, we need only check them all to find the best. Thus our recursive definition for the minimum cost of parenthesiz~~[illegible]~~ing the product $A_i A_{i+1} \ldots A_j$ becomes

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

The $m[i,j]$ values give costs of optimal solutions to subproblems. To help us track of how to construct an optimal solution, [let us define $s[i,j]$ to be a value of k at which we can split the product $A_i A_{i+1} \ldots A_j$ to obtain an optimal parenthesization. That is, $s[i,j]$ equals a value k such that

$$m[i,j] = m[i,k] + m[k+1,j] + P_{i-1} P_k P_j]$$

The following pseudocode assumes that matrix A_i has dimensions $P_{i-1} \times P_i$ for $i = 1, 2 \dots n$. The input is a sequence $P = \langle P_0, P_1, \dots P_n \rangle$, where $length[P] = n+1$. The procedure uses an auxillary table $m[1 \dots n, 1 \dots n]$ for storing the $m[i,j]$ costs and an auxillary table $s[1 \dots n, 1 \dots n]$ that records which index of k achieved the optimal cost in computing $m[i,j]$. We will use the table s to construct an optimal solution.

```
    MATRIX-CHAIN-ORDER (P)
1   n ← length[P] − 1
2   for i ← 1 to n
3       do m[i, i] ← 0
4   for l ← 2 to n          // l is the chain length.
5       do for i ← 1 to n−l+1
6           do j ← i+l−1
7              m[i, j] ← ∞
8              for k ← i to j−1
9                  do q ← m[i, k]+m[k+1, j]+P_{i-1}P_kP_j
10                    if q < m[i, j]
11                      then m[i, j] ← q
12                           S[i, j] ← k
13  return m and s
```

i: 1 to 5 6-2+1 7-2-5

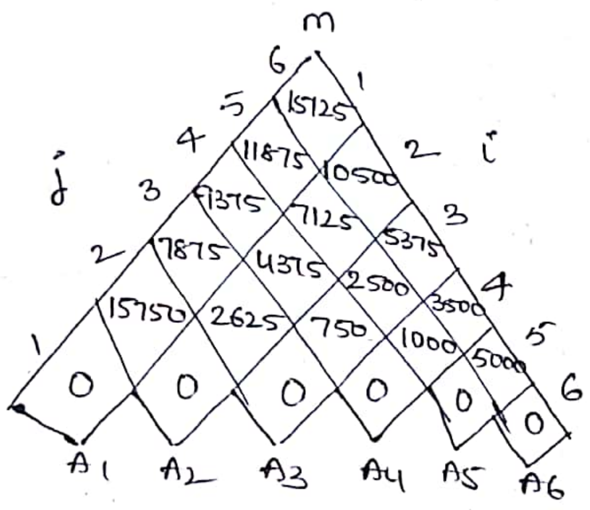
1 2 6-3+1

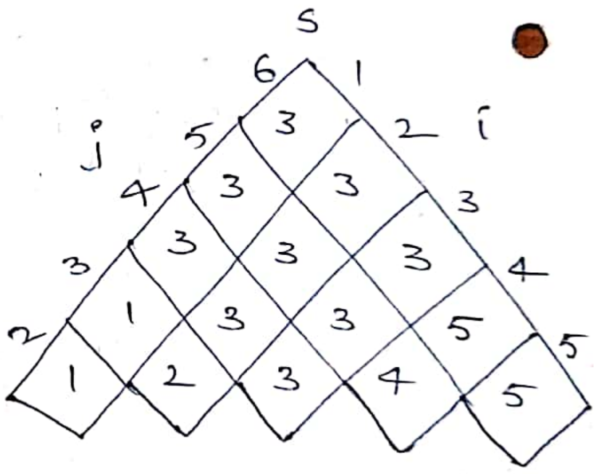
1+2-1 4

2+2-1

The algorithm first computes $m[i, i] \leftarrow 0$ for $i = 1, 2, \dots n$ (the minimum costs for chains of length 1) in lines 2-3. It then uses recurrence to compute $m[i, i+1]$ for $i = 1, 2, \dots n-1$ (the minimum costs for chains of length $l = 2$) during the first execution of the loop in lines 4-12. The second time through the loop, it computes $m[i, i+2]$ for $i=1,2,\ldots n-2$ (the minimum cost for chains of length=3) and so forth. At each step, the $m[i,j]$ cost computed in lines 9-12 depends only on table entries $m[i,k]$ and $m[k+1, j]$ already computed.

The following figure illustrates this procedure on a chain of $n=6$ matrices. Since we have defined $m[i,j]$ only for $i \leq j$, only the portion of the table m strictly above the main diagonal is used.





matrix	dimension
A1	30×35
A2	35×15
A3	15×5
A4	5×10
A5	10×20
A6	20×25

A simple inspection of the nested loop structure of MATRIX-CHAIN ORDER yields a running time of $O(n^3)$ for the algorithm. The loops are nested three deep, and each loop index (l, i, and k) takes almost $n-1$ values. The alg requires $\Theta(n^2)$ space to store m & s tables. Thus, MATRIX-CHAIN-ORDER is much more efficient than the exponential-time method of enumerating all possible parenthesizations and checking each one.

Exercise:-

→ Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is $\langle 5, 10, 3, 12, 5, 50, 6 \rangle$

Solution

$i=j$

$m[1,1]=0$, $m[2,2]=0$, $m[3,3]=0$

$m[4,4]=0$, $m[5,5]=0$, $m[6,6]=0$.

$j=i+1$

$\boxed{k=1}$ ✓

$m[1,2] = m[1,1] + m[2,2] + P_0 P_1 P_2$

$= 0 + 0 + 30\times35\times15$

$= \boxed{15750}$ ✓

$m[2,3] = m[2,2] + m[3,3] + P_1 P_2 P_3$

$= 0 + 0 + 35\times15\times5$

$= \boxed{2625}$ ✓ $\boxed{k=2}$ ✓

$m[3,4] = m[3,3] + m[4,4] + P_2 P_3 P_4$

$= 0 + 0 + 15\times5\times10$

$= \boxed{750}$ ✓ $\boxed{k=3}$ ✓

$m[4,5] = m[4,4] + m[5,5] + P_3 P_4 P_5$

$= 0 + 0 + 5\times10\times20$

$= \boxed{1000}$ ✓ $\boxed{k=4}$ ✓

$m[5,6] = m[5,5] + m[6,6] + P_4 P_5 P_6$

$= 0 + 0 + 10\times20\times25$

$= \boxed{5000}$ ✓ $\boxed{k=5}$ ✓

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

$\langle P_0 \cdots P_6 \rangle = \langle 30, 35, 15, 5, 10, 20, 25 \rangle$

$P_0\ P_1\ P_2\ P_3\ P_4\ P_5\ P_6$

Matrix A_i has dimension $P_{i-1}\times P_i$ for $i=1 \cdots n$.

Solution

$j = i+2$

$K = 1, 2$

$$m[1,3] = \min\begin{cases} m[1,1] + m[2,3] + P_0 P_1 P_3 \\ m[1,2] + m[3,3] + P_0 P_2 P_3 \end{cases}$$

$$= \min\begin{cases} 0 + 2625 + 30 \times 35 \times 5 \\ 15750 + 0 + 30 \times 15 \times 5 \end{cases}$$

$$= \begin{matrix} 2625 + 5250 \\ 15750 + 2250 \end{matrix} \Big\} = \begin{matrix} 7875, \\ 18000 \end{matrix} = \boxed{7875} ✓$$

$\boxed{K=1}$ ✓

$K = 2, 3$

$$m[2,4] = m[2,2] + m[3,4] + P_1 P_2 P_4$$
$$= m[2,3] + m[4,4] + P_1 P_3 P_4$$

$$\min = \begin{cases} 0 + 750 + 35 \times 15 \times 10 \\ 2625 + 0 + 35 \times 5 \times 10 \end{cases}$$

$\boxed{K=3}$ ✓ $\begin{matrix} 750 + 5250 \\ 2625 + 1750 \end{matrix} \Big\} \begin{matrix} 6000, \\ 4375 \end{matrix} = \boxed{4375} ✓$

$K = 3, 4$

$$m[3,5] = m[3,3] + m[4,5] + P_2 P_3 P_5$$
$$= m[3,4] + m[5,5] + P_2 P_4 P_5$$

$$\min\begin{cases} 0 + 1000 + 15 \times 5 \times 20 \\ 750 + 0 + 15 \times 10 \times 20 \end{cases}$$

$$\min\begin{cases} 2500 \\ 3750 \end{cases} = \boxed{2500} ✓ \quad \boxed{K=3} ✓$$

$k=4,5$

$m[4,6] =$

$m[4,4] + m[5,6] + P_3 P_4 P_6$

$m[4,5] + m[6,6] + P_3 P_5 P_6$

$$\min \begin{cases} 0 + 5000 + 5 \times 10 \times 25 \\ 1000 + 0 + 5 \times 20 \times 25 \end{cases}$$

$$\min \begin{cases} 6250 \\ 3500 \end{cases} = \boxed{3500} \checkmark \quad \boxed{k=5} \checkmark$$

$j = i+3$

$k = 1, 2, 3.$

$$m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + P_0 P_1 P_4 \\ m[1,2] + m[3,4] + P_0 P_2 P_4 \\ m[1,3] + m[4,4] + P_0 P_3 P_4 \end{cases}$$

$$= \min \begin{cases} 0 + 4375 + 30 \times 35 \times 10 \\ 15750 + 750 + 30 \times 15 \times 10 \\ 7875 + 0 + 30 \times 5 \times 10 \end{cases}$$

$\boxed{k=3} \checkmark$

$$= \begin{cases} 4375 + 10500 \\ 15750 + 750 + 4500 \\ 7875 + 1500 . \end{cases} = \begin{cases} 14875 \\ 21000 \\ 9375 \end{cases} = \boxed{9375} \checkmark$$

$k = 2, 3, 5$

$m[2,5] =$

$m[2,2] + m[3,5] + P_1 P_2 P_5$

$m[2,3] + m[4,5] + P_1 P_3 P_5$

$m[2,4] + m[5,5] + P_1 P_4 P_5$

$$= \min \begin{cases} 0 + 2500 + 35\times15\times20 \\ 2625 + 1000 + 35\times5\times20 \\ 4375 + 0 + 35\times10\times20 \end{cases}$$

$$= \min \begin{cases} 13000, \\ 7125, \\ 11375 \end{cases} = \boxed{7125} ✓ \quad \boxed{K=3} ✓$$

$K=3,4,5$

$$m[3,6] = \min \begin{cases} m[3,3] + m[4,6] + P_2 P_3 P_6 \\ m[3,4] + m[5,6] + P_2 P_4 P_6 \\ m[3,5] + m[6,6] + P_2 P_5 P_6 \end{cases}$$

375

$$= \min \begin{cases} 0 + 3500 + 15\times5\times25 \\ 750 + 5000 + 15\times10\times25 \\ 2500 + 0 + 15\times20\times25 \end{cases}$$

$$= \min \begin{cases} 5375, \\ 9500, \\ 10,000 \end{cases} = \boxed{5375} ✓ \quad \boxed{K=3} ✓$$

$j = i+4 \qquad K = 1, 2, 3, 4.$

$$m[1,5] = \min \begin{cases} = m[1,1] + m[2,5] + P_0 P_1 P_5 \\ = m[1,2] + m[3,5] + P_0 P_2 P_5 \\ = m[1,3] + m[4,5] + P_0 P_3 P_5 \\ = m[1,4] + m[5,5] + P_0 P_4 P_5 \end{cases}$$

600

$$\min = \begin{cases} 0 + 7125 + 30 \times 35 \times 20 \\ 15750 + 2500 + 30 \times 15 \times 20 \\ 7875 + 1000 + 30 \times 5 \times 20 \\ 9375 + 0 + 30 \times 10 \times 20 \end{cases}$$

$$= \begin{cases} 28125, \\ 27250, \\ 11875, \\ 15,375. \end{cases} = \boxed{11875} \checkmark \boxed{k=3} \checkmark$$

$k = 2, 3, 4, 5$

$$m[2,6] = \min \begin{cases} m[2,2] + m[3,6] + P_1 P_2 P_6 \\ m[2,3] + m[4,6] + P_1 P_3 P_6 \\ m[2,4] + m[5,6] + P_1 P_4 P_6 \\ m[2,5] + m[6,6] + P_1 P_5 P_6 \end{cases}$$

$$= \begin{cases} 0 + 5375 + 35 \times 15 \times 25 \\ 2625 + 3500 + 35 \times 5 \times 25 \\ 4375 + 5000 + 35 \times 10 \times 25 \\ 7125 + 0 + 35 \times 20 \times 25 \end{cases}$$

875

$$= \begin{cases} 18500, \\ 10500, \\ 18125, \\ 24625 \end{cases} = \boxed{10500} \checkmark \boxed{k=3} \checkmark$$

$J = i + 5$ $k = 1, 2, 3, 4, 5.$

$$m[1,6] = \min \begin{cases} m[1,1] + m[2,6] + P_0 P_1 P_6 \\ m[1,2] + m[3,6] + P_0 P_2 P_6 \\ m[1,3] + m[4,6] + P_0 P_3 P_6 \\ m[1,4] + m[5,6] + P_0 P_4 P_6 \\ m[1,5] + m[6,6] + P_0 P_5 P_6 \end{cases}$$

$$\min \begin{cases} 0 + 10500 + 30\times35\times25 \\ 15750 + 5375 + 30\times15\times25 \\ 7875 + 3500 + 30\times5\times25 \\ 9375 + 5000 + 30\times10\times25 \\ 11875 + 0 + 30\times20\times25 \end{cases}$$

750

$$\min \begin{cases} 36750, \\ 32375, \\ 15125, \\ 21875, \\ 26875 \end{cases} = \boxed{15125} \quad \boxed{k=3}$$

$\therefore$ mincost = 15125 scalar multiplications are needed

In m[1,6] we have m[1,3] + m[4,6] for k=3 i.e for min.
So split (A1 A2 A3)(A4 A5 A6). again see in m[1,3] & m[4,6] for splitting.

(A1 (A2 A3)) ((A4 A5) A6)

7875

$\Rightarrow$ ((A1(A2 A3))((A4 A5) A6))

A2×A3
= 2625
(30×35)(35×5)
(30×5)(5×25)

3750

1000
3500

(5×20)(20×25)

(A2 A3) = (35×15)(15×5) ⇒ 35×5
= 35×15×5
= 2625

A1 (A2 A3) = (30×35)(35×5) ⇒ 30×5
= 30×35×5
= 5250

2625 + 5250 = 7875

(A4 A5) = (5×10)(10×20) ⇒ 5×10×20 ⇒ 1000
⇒ 5×20
(A4 A5).A6 = (5×20)(20×25) ⇒ 5×20×25 ⇒ 2500
⇒ 5×25
1000 + 2500 = 3500

⇒ (30×5) (5×25) ⇒ 30×5×25 = 3750

(A1 (A2 A3))((A4 A5) A6) ⇒ 7875 + 3500 + 3750 ⇒ $\boxed{15125}$
Hence Proved.