The Matrix Chain Multiplication Problem: given a chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications. A product of matrices is fully parenthesized if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses.

Example: There are 5 different ways to compute the product $A_1 A_2 A_3 A_4$ (all 5 will give the same result since matrix multiplication is associative):

$(A_1(A_2(A_3A_4)))$

$(A_1((A_2A_3)A_4))$

$((A_1A_2)(A_3A_4))$

$((A_1(A_2A_3))A_4)$

$(((A_1A_2)A_3)A_4)$

```
Matrix-Multiply(A_{p×q}, B_{q×r})
1 let C be a new p×r matrix
2 for i = 1 to p
3     for j = 1 to r
4         c_ij ← 0
5         for k = 1 to q
6             c_ij ← c_ij + a_ik . b_kj
7 return C
```

Matrix-Multiply$(A_{p\times q}, B_{q\times r})$ has complexity $\Theta(pqr)$

Different parenthesization can give different costs.

Example: $\langle A_{1\times 20}, B_{20\times 1}, C_{1\times 10}\rangle$ has two parenthesizations:

$((A_{1\times 20} B_{20\times 1}) C_{1\times 10}) = (D_{1\times 1} C_{1\times 10}) = E_{1\times 10}$

Cost $= 1\times 20\times 1$ Cost $= 1\times 1\times 10$

Total cost $= 1\times 20\times 1 + 1\times 1\times 10 = 30$

$(A_{1\times 20} (B_{20\times 1} C_{1\times 10})) = (A_{1\times 20} F_{20\times 10}) = G_{1\times 10}$

Cost $= 20\times 1\times 10$ Cost $= 1\times 20\times 10$

Total Cost $= 20\times 1\times 10 + 1\times 20\times 10 = 400$

Counting the number of Parenthesizations: Let the number of alternative parenthesizations of a sequence of n matrices be $P(n)$. When $n=1$, we just have one matrix and therefore only one way to fully parenthesize the matrix product. When $n\geq 2$, a fully parenthesized matrix product is the product of two fully parenthesized matrix subproducts, and the split between the two subproducts may occur between the k^{th} and $(k+1)^{th}$ matrices for any $k = 1, 2, \cdots, n-1$. $\Rightarrow$

$$P(n) = 1 \quad \text{if } n = 1$$

$$P(n) = \sum_{k=1}^{n-1} P(k)\, P(n-k) \quad \text{if } n \geq 2$$

We will solve the above recurrence using generating functions. Let $G(x)$ be the generating function for $P(n)$:

$$G(x) = \sum_{i=1}^{\infty} P(i)\, x^i = P(1)\, x + P(2)\, x^2 + P(3)\, x^3 + \cdots + P(i)\, x^i + \cdots .$$

$$\Rightarrow (G(x))^2 = \left(\sum_{i=1}^{\infty} P(i)x^i\right)^2 = \sum_{i=2}^{\infty}\left(\sum_{k=1}^{i-1} P(k)P(i-k)\right)x^i$$

$$= \sum_{i=2}^{\infty} P(i)x^i = G(x) - P(1)x = G(x) - x$$

$$\Rightarrow (G(x))^2 - G(x) + x = 0$$

$$\Rightarrow G(x) = \frac{1 \pm \sqrt{1-4x}}{2}, \quad G(0) = 0 \Rightarrow G(x) = \frac{1-\sqrt{1-4x}}{2}$$

$$= \frac{1}{2} - \frac{1}{2}\left(1 + \frac{\left(\frac{1}{2}\right)}{1!}(-4)x + \frac{\left(\frac{1}{2}\right)\left(-\frac{1}{2}\right)}{2!}(-4)^2x^2 + \frac{\left(\frac{1}{2}\right)\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)}{3!}(-4)^3x^3 + \cdots\right.$$

$$\left. + \frac{\left(\frac{1}{2}\right)\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\cdots\left(-\frac{2i-3}{2}\right)}{i!}(-4)^i x^i + \cdots\right)$$

$$= \sum_{i=1}^{\infty}\left(\frac{1}{2i-1}\right)\binom{2i-1}{i-1}x^i = \sum_{i=1}^{\infty} P(i)x^i$$

$$\Rightarrow P(n) = \frac{1}{2n-1}\binom{2n-1}{n-1} = \frac{(2n-2)!}{n!\,(n-1)!} = \frac{(2n)!\,n}{(n!)^2(2n)(2n-1)}$$

$$> \frac{(2n)!}{(n!)^2(4n)}$$

Now using the formula $\sqrt{2\pi}\,n^{n+\frac{1}{2}}e^{-n} \le n! \le e\,n^{n+\frac{1}{2}}e^{-n}$

we get $P(n) > \dfrac{\sqrt{2\pi}\,(2n)^{2n+\frac{1}{2}}e^{-2n}}{(e\,n^{n+\frac{1}{2}}e^{-n})^2(4n)} = \left(\dfrac{\sqrt{2\pi}}{4e^2}\right)\dfrac{4^n}{n^{3/2}}$

$$\Rightarrow \boxed{P(n) = \Omega\left(\frac{4^n}{n^{3/2}}\right)}$$

A Dynamic Programming Algorithm has four steps:

(1) Find the optimal substructure of the problem
(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: optimal substructure of Parenthesization:

Let the optimal parenthesization of $A_i A_{i+1} \dots A_j$ be

$((A_i \dots A_k)(A_{k+1} \dots A_j))$

→ The two subproblems $< A_i, \dots A_k >$ and $< A_{k+1}, \dots A_j >$ should have optimal parenthesization, otherwise the original problem will not have optimal parenthesization.

Step 2: Recursive solution: Let $m[i,j]$ be the minimum number of scalar multiplications needed to compute the matrix $A_{i \dots j}$.

$$(A_i A_{i+1} \dots A_j) = ((A_i \dots A_k)(A_{k+1} \dots A_j)) = (B_{p_{i-1} \times p_k} (A_{k+1} \dots A_j))$$

cost = $m[i,k]$

$$= (B_{p_{i-1} \times p_k} \; C_{p_k \times p_j}) = D_{p_{i-1} \times p_j}$$

cost = $m[k+1, j]$ cost = $p_{i-1} p_k p_j$

$\Rightarrow$ Total cost = $m[i,k] + m[k+1,j] + p_{i-1} p_k p_j$

$\Rightarrow$ we have to choose best k $\Rightarrow$

$$m[i,j] = 0 \quad \text{if } i = j$$

$$m[i,j] = \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \} \quad \text{if } i < j$$

Step3 : Computing the optimal costs : we observe that optimal matrix chain of length $j-i+1$ in $m[i,j]$ is computed using smaller optimal matrix chains of length $k-i+1$ $(i \le k < j)$ in $m[i,k]$ and of length $j-k$ $(i \le k < j)$ in $m[k+1, j]$.
In our algorithm, we follow the bottom up approach.
First we start with computing the smallest optimal chains of length 1: $\langle A_1 \rangle, \langle A_2 \rangle \cdots \langle A_n \rangle$, then of length 2:

$\langle A_1, A_2 \rangle, \langle A_2, A_3 \rangle, \ldots \langle A_{n-1}, A_n \rangle$ and so on

upto chain of length n: $\langle A_1, A_2, \ldots, A_n \rangle$.

```
Matrix-Chain-Order (< p0, p1, ..., pn>)
1  let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
2  for i=1 to n
3      m[i,i] ← 0
4  for l = 2 to n          // l is the chain length
5      for i = 1 to n-l+1
6          j ← i+l-1
7          m[i,j] ← ∞
8          for k = i to j-1
9              q ← m[i,k] + m[k+1,j] + pi-1 pk pj
10             if q < m[i,j]
11                 m[i,j] ← q
12                 s[i,j] ← k
13 return m and s
```

Step 4: Each entry $S[1..n-1, 2..n]$ records a value of k such that an optimal parenthesization of $A_i \cdots A_j$ splits the product between A_k and A_{k+1}.

$\Rightarrow A_{1..n} = (A_{1..S[1,n]} \; A_{S[1,n]+1..n})$ and we repeat the recursion

```
Print-Optimal-Parens(S, i, j)
1   if i == j
2       print "(Ai)"
3   else print "("
4       Print-Optimal-Parens(S, i, S[i,j])
5       Print-Optimal-Parens(S, S[i,j]+1, j)
6       Print ")"
```

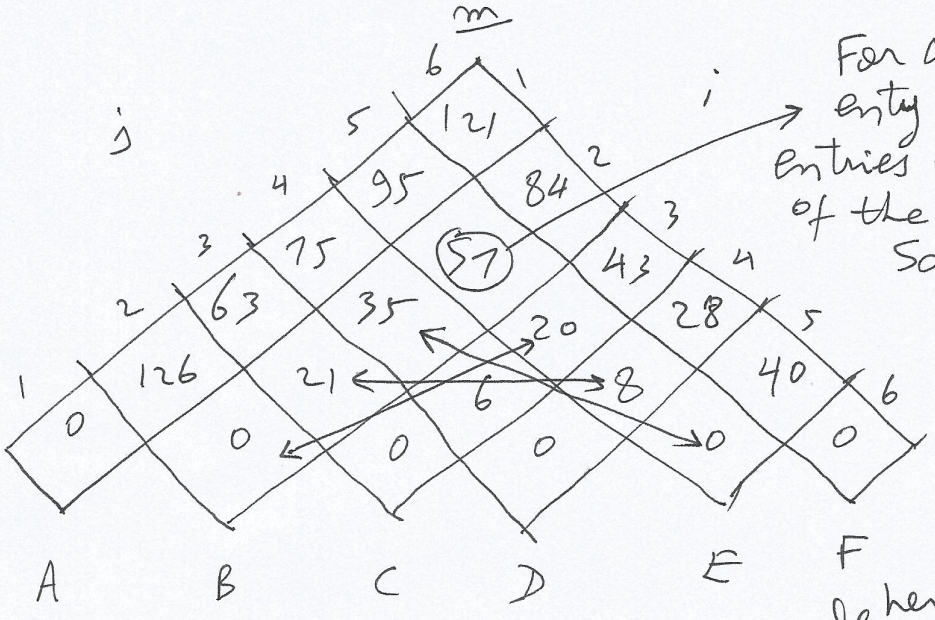
Complexity of Matrix Chain Order: Complexity is dominated by the for loop from lines 4 to 12. The for loop of lines 8 to 12 has complexity $\theta(j-i) = \theta(l-1)$. The for loop of line 5 runs for $n-l+1$ times contributing $\theta((n-l+1)(l-1))$. For loop of line 4 runs from $l=2$ to n.

$$\Rightarrow T(n) = \theta\left(\sum_{l=2}^{n} (n-l+1)(l-1)\right) = \theta\left(\sum_{l'=1}^{n-1} (n-l')\,l'\right)$$

$$= \theta\left(n\sum_{l'=1}^{n-1} l' - \sum_{l'=1}^{n-1} l'^2\right) = \theta\left(\frac{n(n-1)n}{2} - \frac{(n-1)n(2n-1)}{6}\right)$$

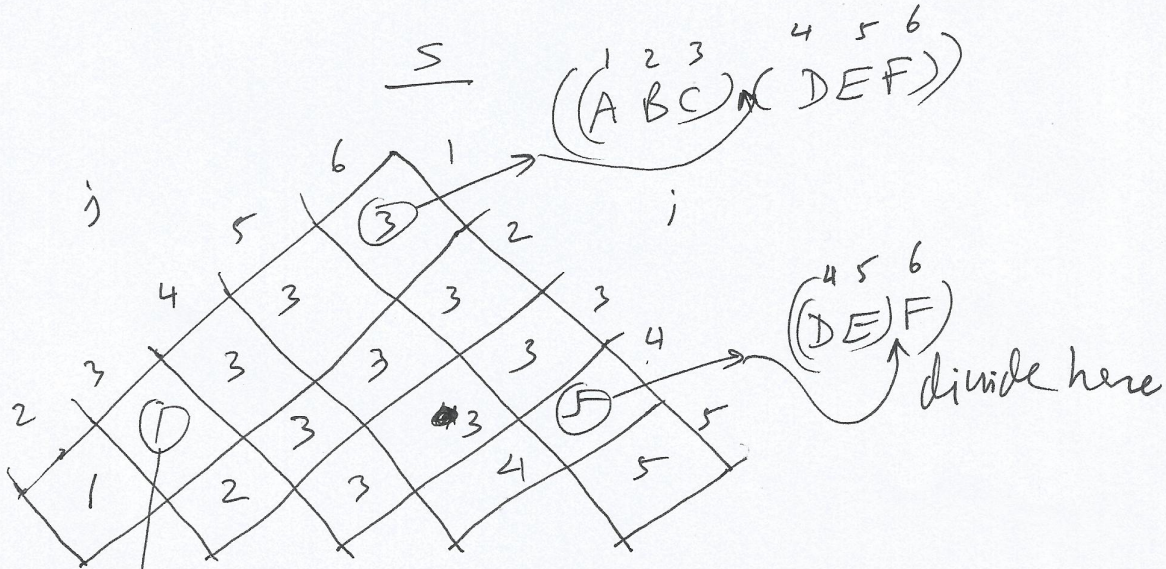
$$= \boxed{\theta(n^3)}$$

Example: $< A_{6\times7}, B_{7\times3}, C_{3\times1}, D_{1\times2}, E_{2\times4}, F_{4\times5} >$



For computing this entry we use the entries at the end of the arrows. Same pattern is followed in each entry.

Divide here



((A(BC))((DE)F))