MATRIX CHAIN MULTIPLICATION

→ Suppose we have a collection of n two dimensional matrices for which we wish to compute the product.

$$A = A_1 \cdot A_2 \cdot A_3 \cdots A_n$$

→ We will evaluate product of n matrices by using standard algorithm for multiplying pair of matrices to resolve all ambiguities in how the matrices are multiplied together.

→ Matrix multiplication is associative in nature.

$$A_1(A_2 \cdot A_3) = (A_1 \cdot A_2) A_3$$

→ Example: chain of matrices is $\langle A_1, A_2, A_3, A_4 \rangle$

The product of $A_1 \cdot A_2 \cdot A_3 \cdot A_4$ can be fully parenthesized in five ways.

$(A_1(A_2(A_3 \cdot A_4)))$

$(A_1 \cdot ((A_2 \cdot A_3) \cdot A_4))$

$((A_1 \cdot A_2)(A_3 \cdot A_4))$

$((A_1 \cdot (A_2 \cdot A_3))A_4)$

$(((A_1 \cdot A_2) \cdot A_3)A_4)$

```
MATRIX-MULTIPLICATION(A, B)
if columns[A] ≠ rows[B]
   then error "Incompatible dimensions".
   else for i ← 1 to rows[A]
           do for j ← 1 to columns[B]
                 do c[i,j] ← 0
                    for k ← 1 to columns[A]
                        do c[i,j] ← c[i,j] + A[i,k] · B[k,j]
        return c
```

→ The two matrices A and B can be multiplied if they are compatible, i.e. the number of columns of A must be equal to the number of rows of B.

→ If matrix A having dimension $p \times q$ and matrix B having dimension $q \times r$, then the resulting matrix C is $p \times r$.

→ The time to compute the number of scalar multiplications of matrix C is pqr.

→ Consider 3 matrices ⟨A, B, C⟩ having size $5 \times 10, 10 \times 50, 50 \times 20$.

→ We will parenthesize these 3 matrices either $A \cdot (B \cdot C)$ or $(A \cdot B) \cdot C$

→ Computing $A \cdot (B \cdot C)$ requires

$$5 \cdot 10 \cdot 20 + 10 \cdot 50 \cdot 20$$

$$= 1000 + 10000 = 11000 \text{ multiplications}$$

→ Computing $(A \cdot B) \cdot C$ requires

~~5·10·20 + 10·50~~

~~(5×10, 10×50) · 50×20~~

$$5 \cdot 10 \cdot 50 + 5 \cdot 50 \cdot 20 = 2500 + 5000 = 7500 \text{ multiplications}$$

→ In matrix multiplication problem, we are not actually multiplying matrices. Our objective is only to determine an order by which matrices are multiplied at lowest cost.

→ One way to solve matrix multiplication problem is to simply enumerate all the possible ways of parenthesization of the ~~expression~~ set of matrices. and determine the number of multiplications performed by each one

→ Let $T(n)$ be the number of alternatives parenthesizations of a sequence of n matrices.

→ When $n=1$, there is just 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 ~~subproblem~~ subproducts and the split between the two subproblems may occur between the kth and $(k+1)$st matrices for any $k = 1, 2, 3, \cdots, n-1$.

→ Thus, the recurrence is

$$\boxed{T(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} T(k)\,T(n-k) & \text{if } n \geq 2 \end{cases}}$$

Step 1: Determine the structure of an optimal parenthesization

→ 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 $i \leq k < j$.

→ For some value of k, we first compute the matrices $A_{i..k}$ and $A_{k+1..j}$. 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 $A_{i..k}$ + the cost of computing $A_{k+1..j}$ + the cost of multiplying them together.

Step 2: A recursive solution

→ For matrix chain multiplication problem, we have to determine the minimum cost of a parenthesization of $A_i A_{i+1} \cdots A_j$ for $1 \leq i \leq j \leq n$.

→ Let $m[i, j]$ be the minimum number of scalar multiplications needed to compute the matrix $A_{i..j}$

→ For the full problem, the cost of a cheapest way to compute $A_{1..n}$ would be $m[1, n]$.

→ We can define $m[i, j]$ recursively as follows:
If $i = j$, the problem is trivial i.e. the chain consists of only one matrix. So no scalar multiplications are necessary to compute the product.

→ Thus $m[i, i] = 0$, for $i = 1, 2, \cdots n$.
If $i \neq j$, we have to take the structure of an optimal 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 $i \leq k < j$.

→ ∴ $m[i, j]$ is equal to the minimum cost for computing the sub products $(A_i \cdots A_k)$ and $(A_{k+1} \cdots A_j)$ plus the cost of multiplying these two matrices together.

→ Each matrix A_i is $P_{i-1} \times P_i$ scalar multiplications. Computing the matrix product $A_{i..k} \cdot A_{k+1..j}$ takes $P_{i-1} \cdot P_k \cdot P_j$ scalar multiplications.

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

→ Here we assume that the value of K is known to us, but in really we donot know the value of K. The possible value of K is from i to $j-1$.

→ It means, there are only $j-i$ possible values for K. Since the optimal parenthesization must use one of these values for K, we need only check them all to find the best.

→ Thus the recursive defination for the minimum cost of parenthesizing the product $A_i A_{i+1} \cdots A_j$ becomes:

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

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

→ To keep 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 $A_i A_{i+1} \cdots A_j$ to obtain an optimal parenthesization.

→ $S[i,j]$ equals a value K such that
$$m[i,j] = m[i,K] + m[K+1,j] + P_{i-1} P_K P_j$$

Step 3: Computing the optimal costs in a bottom up fashion.

→ A recursive algorithm may encounter each subproblem many times in different branches of its recursion tree. This property of overlapping subproblems is the 2nd feature of dynamic programming (First feature is optimal substructure)

→ Now we will calculate the optimal cost by using a tabular, bottom up approach.

→ A_i has dimensions $P_{i-1} \times P_i$ for $i = 1, 2, \ldots, n$

The input sequence is $P = \langle P_0, P_1, \ldots, P_n \rangle$ where $\text{length}[P] = n+1$.

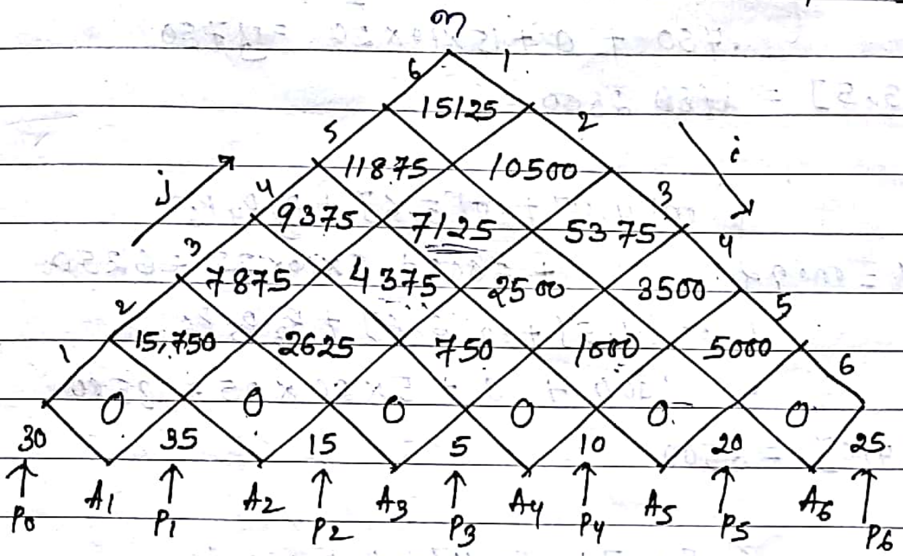
→ An auxilary table $m[1..n, 1..n]$ for storing the $m[i,j]$ costs and an auxilary table $S[1..n, 1..n]$ that records which index of K achieved the optimal cost in computing $m[i,j]$. We will use the table S to construct an optimal solution.

MATRIX-CHAIN-ORDER(P)

```
n ← length[P] − 1
for i ← 1 to n
        do m[i, i] ← 0
for l ← 2 to n            ▷ l is the chain length
        do for i ← 1 to n−l+1
                do j ← i+l−1
                   m[i, j] ← ∞
                   for k ← i to j−1
                        do q ← m[i, k] + m[k+1, j] + P_{i−1} P_k P_j
                           if q < m[i, j]
                              then m[i, j] ← q
                                   S[i, j] ← k
return m and s
```

Detail soln: $(30\times35)(35\times15)(15\times5)(5\times10)(10\times20)(20\times25)$

30	35	15	5	10	20	25
P_0	P_1	P_2	P_3	P_4	P_5	P_6



$m[1,1] = m[2,2] = m[3,3] = m[4,4] = m[5,5] = m[6,6] = 0$

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

$= 0 + 0 + 30\times35\times15 = 15,750.$

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

$= 0 + 0 + 35\times15\times5 = 2625$

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

$= 0 + 0 + 750 = 750$

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

$= 0 + 0 + 5\times10\times20 = 1000$

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

$= 0 + 0 + 10\times20\times25 = 5000$

$$m[1,3] = \min\begin{cases} m[1,1] + m[2,3] + P_0 P_1 P_3 \\ 0 + 2625 + 30\times35\times5 = 7875 \\ m[1,2] + m[3,3] + P_0 P_2 P_3 \\ 15,750 + 0 + 30\times15\times5 = 18000 \end{cases}$$

$\therefore m[1,3] = 7875$

$$m[2,4] = \min\begin{cases} m[2,2] + m[3,4] + P_1 P_2 P_4 \\ 0 + 750 + 35\times15\times10 = 6000 \checkmark \\ m[2,3] + m[4,4] + P_1 P_3 P_4 \\ 2625 + 0 + 35\times5\times10 = \cancel{9125}\ 4375 \text{ (Ans)} \end{cases}$$

$\therefore m[2,4] = \cancel{9125}\ 4375$

$$m[3,5] = \min \begin{cases} m[3,3] + m[4,5] + P_2 P_3 P_5 \\ 0 + 1000 + 15 \times 5 \times 20 = 2500 \\ m[3,4] + m[5,5] + P_2 P_4 P_5 \\ 750 + 0 + 15 \times 10 \times 20 = 3750 \end{cases}$$

$\therefore m[3,5] = \cancel{1750}\ 2500$

$$m[4,6] = \min \begin{cases} m[4,4] + m[5,6] + P_3 P_4 P_6 \\ 0 + 5000 + 5 \times 10 \times 25 = 6250 \\ m[4,5] + m[6,6] + P_3 P_5 P_6 \\ 1000 + 0 + 5 \times 20 \times 25 = 3500 \end{cases}$$

$\therefore m[4,6] = 3500$

$$m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + P_0 P_1 P_4 \\ 0 + 4375 + 30 \times 35 \times 10 = 14875 \\ m[1,2] + m[3,4] + P_0 P_2 P_4 \\ 15750 + 750 + 30 \times 15 \times 10 = 21000 \\ m[1,3] + m[4,4] + P_0 P_3 P_4 \\ 7875 + 0 + 30 \times 5 \times 10 = 9375 \end{cases}$$

$\therefore m[1,4] = 9375$

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + P_1 P_2 P_5 \\ 0 + 2500 + 35 \times 15 \times 20 = 13000 \\ m[2,3] + m[4,5] + P_1 P_3 P_5 \\ 2625 + 1000 + 35 \times 5 \times 20 = 7125 \\ m[2,4] + m[5,5] + P_1 P_4 P_5 \\ 4375 + 0 + 35 \times 10 \times 20 = 11375 \end{cases}$$

$\therefore m[2,5] = 7125$

$$m[3,6] = \min \begin{cases} m[3,3] + m[4,6] + P_2 P_3 P_6 \\ 0 + 3500 + 15 \times 5 \times 25 = 5375 \\ m[3,4] + m[5,6] + P_2 P_4 P_6 \\ 750 + 5000 + 15 \times 10 \times 25 = 9500 \\ m[3,5] + m[6,6] + P_2 P_5 P_6 \\ 2500 + 0 + 15 \times 20 \times 25 = 10000 \end{cases}$$

$\therefore m[3,6] = 5375$

$$m[1,5] = \min \begin{cases} m[1,1] + m[2,5] + p_0 p_1 p_5 \\ 0 + 7125 + 30 \times 35 \times 20 = 28125 \\ m[1,2] + m[3,5] + p_0 p_2 p_5 \\ 15750 + 2500 + 30 \times 15 \times 20 = 27250 \\ m[1,3] + m[4,5] + p_0 p_3 p_5 \\ 7875 + 1000 + 30 \times 5 \times 20 = 11875 \\ m[1,4] + m[5,5] + p_0 p_4 p_5 \\ 9375 + 0 + 30 \times 10 \times 20 = 15375 \end{cases}$$

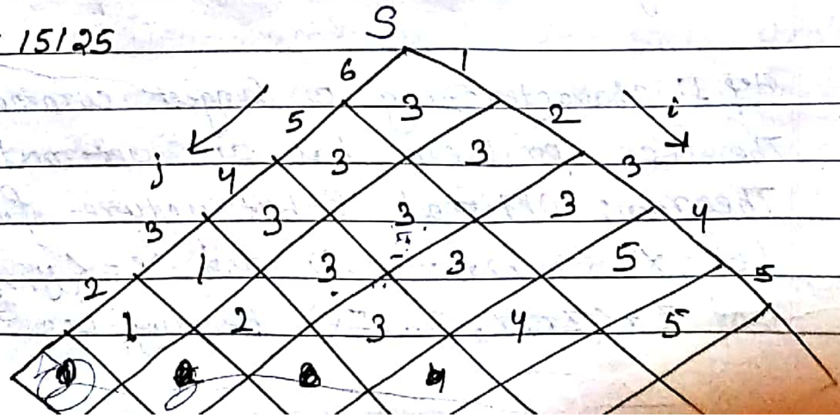
$\therefore m[1,5] = 11875$

$$m[2,6] = \min \begin{cases} m[2,2] + m[3,6] + p_1 p_2 p_6 \\ 0 + 5375 + 35 \times 15 \times 25 = 18500 \\ m[2,3] + m[4,6] + p_1 p_3 p_6 \\ 2625 + 3500 + 35 \times 5 \times 25 = 10500 \\ m[2,4] + m[5,6] + p_1 p_4 p_6 \\ 4375 + 5000 + 35 \times 10 \times 25 = 18125 \\ m[2,5] + m[6,6] + p_1 p_5 p_6 \\ 7125 + 0 + 35 \times 20 \times 25 = 24625 \end{cases}$$

$\therefore m[2,6] = 10500$

$$m[1,6] = \min \begin{cases} m[1,1] + m[2,6] + p_0 p_1 p_6 \\ 0 + 10500 + 30 \times 35 \times 25 = 36750 \\ m[1,2] + m[3,6] + p_0 p_2 p_6 \\ 15750 + 5375 + 30 \times 15 \times 25 = 32375 \\ m[1,3] + m[4,6] + p_0 p_3 p_6 \\ 7875 + 3500 + 30 \times 5 \times 25 = 15125 \\ m[1,4] + m[5,6] + p_0 p_4 p_6 \\ 9375 + 5000 + 30 \times 10 \times 25 = 21875 \\ m[1,5] + m[6,6] + p_0 p_5 p_6 \\ 11875 + 0 + 30 \times 20 \times 25 = 26875 \end{cases}$$

$\therefore m[1,6] = 15125$



Step 4: Constructing an optimal solution

→ This determines the optimal number of scalar multiplications needed to ~~compu~~ compute a matrix chain product, it doesnot directly show how to multiply the matrices.

→ we are maintaining an array ~~S[1..n]~~ S[1..n, 1..n] where S[i,j] denotes K for the optimal splitting in computing $A_{i..j} = A_{j..k} \, A_{k+1..j}$

→ The array S[1..n, 1..n] can be used recursively to recover the multiplication sequence.

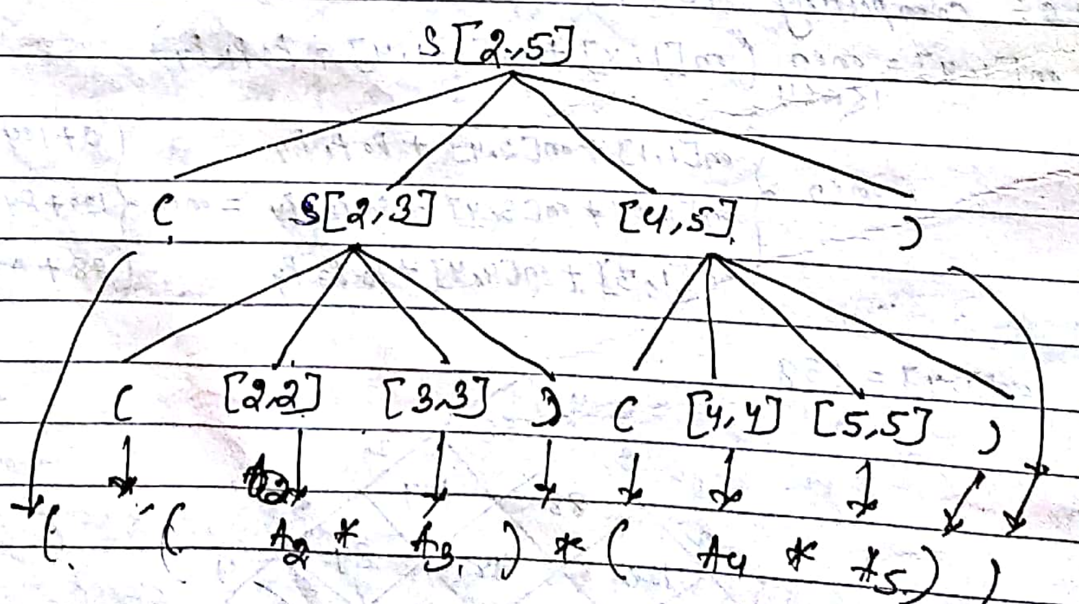
$$S[1,n] = (A_1 \cdots A_{S[1,n]})(A_{S[1,n]+1} \cdots A_n)$$

$$S[1, S[1,n]] = (A_1 \cdots A_{S[1,S[1,n]]})(A_{S[1,S[1,n]]+1} \cdots A_{S[1,n]})$$

$$\vdots \qquad \vdots \qquad \vdots$$

we can do it recursively until the multiplication sequence is determined.

```
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 ")"
```



$((A_2 * A_3) * (A_4 * A_5))$