To develop a dynamic programming solution, we consider the general subproblem of finding an optimal parenthesization of $A_i \ldots A_j$ where $1 \leq i \leq j \leq n$.

Observe that an optimal parenthesization splits $A_i \ldots A_j$ into

$$(A_i \ldots A_k)(A_k \ldots A_j)$$

for some $k$ ($i \leq k < j$).

Note also that if $A_i \ldots A_j$ is optimally parenthesized, then so are both $(A_i \ldots A_k)$ and $(A_k \ldots A_j)$.

Proof: if the parenthesization of $(A_i \ldots A_k)$ is not optimal, then we can replace it with an optimal one yielding a parenthesization of $A_i \ldots A_k$ with fewer scalar multiplications. This contradicts that our original parenthesization of $A_i \ldots A_k$ was optimal.

Thus the principle of optimality is satisfied in this problem.
Let $M_{i,j}$ denote the minimum number of scalar multiplications necessary to compute $A_i \cdots A_j$. If $i = j$ then $M_{i,j} = 0$ since the product consists of just one matrix.

If $i < k < j$, and $k$ is the split position of an optimal parenthesization then

$$M_{i,j} = M_{i,k} + M_{k+1,j} + P_{i-1} P_k P_j$$

Since we don't know $k$, we define in general

$$M_{i,j} = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \left( M_{i,k} + M_{k+1,j} + P_{i-1} P_k P_j \right) & i < j \end{cases}$$
Ex. \( n = 5 \), \( p_0 = 10, p_1 = 20, p_2 = 30, p_3 = 10, p_4 = 40, p_5 = 10 \)

<table>
<thead>
<tr>
<th>Table ( m )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>6000</td>
<td>8000</td>
<td>12000</td>
<td>13000</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>0</td>
<td>6000</td>
<td>14000</td>
<td>12000</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>12000</td>
<td>7000</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>4000</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
</tr>
</tbody>
</table>

For instance, to compute \( m[2,5] \):

\[
m[2,5] = \min \begin{cases} 
m[2,2] + m[3,5] + p_1 p_2 p_3 = 12000 & (k = 2) \\
m[2,3] + m[4,5] + p_1 p_3 p_4 = 12000 & (k = 3) \\
m[2,4] + m[5,5] + p_1 p_4 p_5 = 22000 & (k = 4)
\end{cases}
\]

Observe that to compute \( m[2,5] \) one needs entries \( m[2,2:4] \) and \( m[2..5, 5] \). Thus, to fill the table, first initialize the main diagonal to \( 0 \), then successively fill each off diagonal above the main.

\[ m[i,i] = 0 \quad (1 \leq i \leq n) \]

Then \( m[i, i+1] \quad (1 \leq i \leq n-1) \)

\[ m[i, i+2] \quad (1 \leq i \leq n-2) \]

And in general

\[ m[i, i+l] \quad (1 \leq i \leq n-l) \]

For \( l = 1, 2, \ldots, n-1 \)
See p. 336 for pseudo-code.

From this table we can reconstruct the values $k_i$ which give the split points for each subproblem.

A more efficient approach is to store $k$ values in a parallel table $s[i,j]$ as we construct $m[i,j]$ (p. 336).

By same as above, we see $s[2,5] = 3$, and

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>x</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

From this table we can construct the optimal parameterization:

$$(A_1 (A_2 A_3)) (A_4 A_5)$$
Consider a directed graph in which a weight (cost) is assigned to each directed edge.

We write $G = (V, E)$ where $V$ is the vertex set and $E$ is the set of directed edges. The adjacency matrix of $G$ is defined as $W = (w_{ij})$ where

$$w_{ij} = \begin{cases} 
0 & \text{if } i = j \\
\text{weight of directed edge } (i, j) & \text{if } i \neq j, (i, j) \in E \\
\infty & \text{if } i \neq j, (i, j) \notin E 
\end{cases}$$

Ex.

\[V = \{1, 2, 3, 4\}\]
\[E = \{(1, 2), (2, 3), (4, 1), (4, 3), (3, 2), (2, 2)\}\]

\[W = \begin{pmatrix}
0 & 1 & \infty & \infty \\
\infty & 0 & 1 & 1 \\
\infty & 2 & 0 & \infty \\
2 & \infty & 1 & 0
\end{pmatrix}\]
The weight of a directed \( i-j \) path \((i,j \in V)\) is the sum of the weights of each of its directed edges.

**Problem: (APSP)**

For each pair \((i,j) \in V \times V\), determine an \(i-j\) path of minimum weight. (Also called a shortest path.)

Again there are really two problems:

- Determine the minimum path weights for each \((i,j)\).
- Determine shortest \(i-j\) paths.

We concentrate on the first problem, leaving the second as an exercise.

**Floyd–Warshall Algorithm**

An intermediate vertex of a directed path \( P = (v_1, v_2, \ldots, v_k) \) is any vertex other than \( v_1 \) or \( v_k \), i.e., one of the vertices \( \{v_2, \ldots, v_{k-1}\} \).
Let $G = (V,E)$ be a directed graph with $V = \{1, 2, \ldots, n\}$. Define subsets $V_k$ of $V$ as follows:

$$V_k = \begin{cases} 
\emptyset & k = 0 \\
\{1, 2, \ldots, k\} & 1 \leq k \leq n
\end{cases}$$

Let $(i,j) \in V \times V$ and $1 \leq k \leq n$. Let $P$ denote a minimum weight path among all $i$-$j$ paths with intermediate vertices in $V_k$.

Now observe that we have two alternatives:

- **$k$ is not an intermediate vertex of $P$.** In this case, $P$ is also a minimum weight among all $i$-$j$ paths with intermediate vertices in $V_{k-1}$.

- **$k$ is an intermediate vertex of $P$.** We can decompose $P$ into subpaths $P_1$ and $P_2$:

  $i \longrightarrow P_1 \longrightarrow k \longrightarrow P_2 \longrightarrow j$

Note vertex $k$ is not intermediate to either $P_1$ or $P_2$. 
Thus $P_1$ has minimum weight amongst all $i-k$ paths with intermediate vertices in $V_{k-1}$, and likewise $P_2$ has minimum weight amongst all $k-i$ paths with intermediate vertices in $V_{k-1}$.

These observations show ADSP exhibits optimal substructure, necessary for Dynamic Programming.

Let $d_{ij}^{(k)}$ denote the weight of a minimum weight $i-j$ path with all intermediate vertices in $V_k$.

When $k = 0$, such a path has no intermediate vertices, hence at most one edge. Thus $d_{ij}^{(0)} = w_{ij}$.

The above observations show that for $1 < k$, we have

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

Let $D^{(k)}$ denote the matrix $(d_{ij}^{(k)})$. Then we seek $D^{(n)}$ given $D^{(0)} = W$. 
Floyd-Warshall (W)

1. \( n \leftarrow \text{row}[W] \)
2. \( D^{(0)} \leftarrow W \)
3. for \( k = 1 \) to \( n \)
4. for \( i = 1 \) to \( n \)
5. for \( j = 1 \) to \( n \)
6. \( d^{(k)}_{ij} \leftarrow \min(d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj}) \)
7. return \( D^{(n)} \)

Since (6) takes time \( O(n^3) \), Floyd-Warshall runs in time \( O(n^3) \).

Note that the above algorithm also uses memory \( n^3 \). It is possible to accomplish this with just \( n^2 \) memory (exercise).

To construct shortest paths, we could use \( D = D^{(n)} \) to determine the predecessor matrix \( P^{(n)} = (\pi^{(n)}_{ij}) \), where

\[ \pi^{(n)}_{ij} = \text{predecessor of } i \text{ along a shortest } i-j \text{ path} \]

Alternatively, we could determine intermediate predecessor matrices \( P^{(k)} = (\pi^{(k)}_{ij})(0 \leq k \leq n) \)

\[ \pi^{(k)}_{ij} = \text{predecessor of } i \text{ along a shortest } i-j \text{ path amongst those with intermediate vertex in } V_k. \]
(See p. 632 for details.)

**Exercise**

- Run Floyd-Warshall on the weighted digraph in preceding example.
- Write an algorithm to determine $T$ from $D = D^{(n)}$.
- Alter Floyd-Warshall to build $T^{(k+1)} (0 \leq k \leq n)$ as you go.
- Write an algorithm to print a shortest $i-j$ path given $T = T^{(n)}$.

**Read**

- Longest Common Subsequence (15.4)
- Optimal Binary Search Trees (15.5)