See p. 336 for pseudo-code.

From this table we can re-construct the values $c$, 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).

Ex. Same as above. We see $S[2,5] = 3$, and

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

From this table we can construct the optimal parameterization:

$$\left( A_1, (A_2 A_3) \right) (A_4 A_5)$$
All Pairs Shortest Paths (APSP)  (25.2)

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.

![Graph diagram]

$V = \{1, 2, 3, 4\}$

$E = \{(1, 2), (2, 4), (4, 1), (4, 3), (3, 2), (2, 3)\}$

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

**Problem: (AESP)**

For each pair (i, j) ∈ V × 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 = (v₁, v₂, ..., vₑ) is any vertex other than v₁ or vₑ, i.e., one of the vertices \{v₂, ..., vₑ−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 \( i \rightarrow j \) 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 \( i \rightarrow j \) paths with intermediate vertices in \( V_{k-1} \).

- \( k \) is an intermediate vertex of \( P \). We can decompose \( P \) into subpaths \( P_1 \) and \( P_2 \):

\[
P_1 \rightarrow K \rightarrow P_2
\]

Note vertex \( K \) is not intermediate to either \( P_1 \) or \( P_2 \).
Thus $p_2$ has minimum weight amongst all $i$-$k$ paths with intermediate vertices in $V_{k-1}$, and likewise $p_1$ 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 $c_{ii}^{(k)}$ denote the weight of a minimum weight $i$-$i$ path with all intermediate vertices in $V_k$.

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

The above observations show that for $k \geq 1$ we have

$$c_{ij}^{(k)} = \min(c_{ij}^{(k-1)}, c_{i,k}^{(k-1)} + c_{kJ}^{(k-1)})$$

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

**Floyd-Warshall** (W)

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

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

Note 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 \( \Pi = (\pi_{ij}) \), where

\[
\pi_{ij} = \text{Predecessor of } i \text{ along a shortest } i-j \text{ path}
\]

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

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

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

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