Let $G = (V, E)$ be a directed graph with $\text{cost}(i, j)$ being the cost of the edge $ij$. We want to compute a shortest path between every ordered pairs of vertices.

**NOTATION:** Assume $V = \{1, 2, \ldots, n\}$

For any $1 \leq k \leq n$, let $A^k_{ij}$ denote the cost of a shortest path from $i$ to $j$, so that any intermediate vertex is picked from the set $\{1, 2, \ldots, k\}$

Define $A^0(i, j) = \text{cost}(i, j)$ for any $i, j \in V$.

Recurrence relation is

$$A^k(i, j) = \min \{ A^{k-1}(i, j), A^{k-1}(i, k) + A^{k-1}(k, j) \}$$
Algorithm (outline)
We Generate matrices $A^0, A^1, ..., A^n$ in that order. Note that $A^0$ is the cost matrix. While generating $A^k, k = 1, 2, ..., n$, we compute $A^k(i, j)$ using our recurrence relation in $O(1)$ time. It follows that $A_k$, $k = 1, 2, ..., n$ can be computed in $O(n^2)$, and hence this algorithm has $O(n^3)$ time complexity, and uses $O(n^3)$ space. The following well known version of Floyd-Warshall’s Algorithm uses $O(n^2)$ storage, only.

$A \leftarrow \text{CostMatrix}$
For $k \leftarrow 1$ to $n$ do
For $i \leftarrow 1$ to $n$ do
For $j \leftarrow 1$ to $n$ do
$A(i, j) \leftarrow \min \{ A(i, j), A(i, k) + A(k, j) \}$
\[
D^{(0)} = \begin{pmatrix}
0 & 3 & 8 & \infty & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & \infty & \infty \\
2 & \infty & -5 & 0 & \infty \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\]
\[
\Pi^{(0)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\
4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]
\[
D^{(1)} = \begin{pmatrix}
0 & 3 & 8 & \infty & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & \infty & \infty \\
2 & 5 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\]
\[
\Pi^{(1)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\
4 & 1 & 1 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]
\[
D^{(2)} = \begin{pmatrix}
0 & 3 & 8 & 4 & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & 5 & 11 \\
2 & 5 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\]
\[
\Pi^{(2)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & 2 & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & 2 & 2 \\
4 & 1 & 1 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]
\[
D^{(3)} = \begin{pmatrix}
0 & 3 & 8 & 4 & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & 5 & 11 \\
2 & -1 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\]
\[
\Pi^{(3)} = \begin{pmatrix}
\text{NIL} & 1 & 1 & 2 & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
\text{NIL} & 3 & \text{NIL} & 2 & 2 \\
4 & 3 & 4 & \text{NIL} & 1 \\
\text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL}
\end{pmatrix}
\]
\[
D^{(4)} = \begin{pmatrix}
0 & 3 & 1 & 4 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 3 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 0
\end{pmatrix}
\]
\[
\Pi^{(4)} = \begin{pmatrix}
\text{NIL} & 1 & 4 & 2 & 1 \\
4 & \text{NIL} & 4 & 2 & 1 \\
4 & 3 & \text{NIL} & 2 & 1 \\
4 & 3 & 4 & \text{NIL} & 1 \\
4 & 3 & 4 & 5 & \text{NIL}
\end{pmatrix}
\]
\[
D^{(5)} = \begin{pmatrix}
0 & 1 & 3 & 2 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 3 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 0
\end{pmatrix}
\]
\[
\Pi^{(5)} = \begin{pmatrix}
\text{NIL} & 3 & 4 & 5 & 1 \\
4 & \text{NIL} & 4 & 2 & 1 \\
4 & 3 & \text{NIL} & 2 & 1 \\
4 & 3 & 4 & \text{NIL} & 1 \\
4 & 3 & 4 & 5 & \text{NIL}
\end{pmatrix}
\]
Matrix Chain Problem

We are given matrices $A_1, A_2, A_3, ..., A_n$, $A_i$ has $p_{i-1}$ rows and $p_i$ columns. We want to compute

$$A_1 \times A_2 \times .... \times A_n$$

The problem is to find an order in which we can do the multiplications so that the number of scalar multiplications is minimized.

NOTATIONS:

For $j \geq i$, let $m[i, j]$ denote the optimal answer for the matrices $A_i \ldots A_j$. We define $m[i, i] = 0$. Note that we need to compute $m[1, n]$.

For $j > i$ we have the following recurrence relation

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1}p_kp_j \}$$
Matrix Chain Problem

Algorithm (outline)
Our Algorithms has n stages. At stage $l = 0, 1, ..., n - 1$, we generate all $m[i, j]$, with $j - i = l$ and store them. (m is stored as a 2 dimensional array.) Thus, we first compute and store all $m[i, j]$ with $j - i = 0$, then compute and store all $m[i, j]$ with $j - i = 1$, etc. When computing $m[i, j]$ at stage $l$ (that is $j - i = l$) we use the recurrence relation. Then each $m[i, j]$ can be computed in $O(l)$ or time $O(n)$ time. Since we need to compute $O(n^2)$ entries for m, the time complexity is $O(n^3)$. Note that a tighter analysis gives $\Theta(n^3)$ time complexity.
The $m$ and $s$ tables computed by \textsc{Matrix-Chain-Order} for $n = 6$ and the following matrix dimensions:

<table>
<thead>
<tr>
<th>matrix</th>
<th>dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>$30 \times 35$</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$35 \times 15$</td>
</tr>
<tr>
<td>$A_3$</td>
<td>$15 \times 5$</td>
</tr>
<tr>
<td>$A_4$</td>
<td>$5 \times 10$</td>
</tr>
<tr>
<td>$A_5$</td>
<td>$10 \times 20$</td>
</tr>
<tr>
<td>$A_6$</td>
<td>$20 \times 25$</td>
</tr>
</tbody>
</table>

The tables are rotated so that the main diagonal runs horizontally. Only the main diagonal and upper triangle are used in the $m$ table, and only the upper triangle is used in the $s$ table. The minimum number of scalar multiplications to multiply the 6 matrices is $m[1, 6] = 15,125$. Of the darker entries, the pairs that have the same shading are taken together in line 9 when computing

\[
m[2, 2] + m[3, 5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13000, \\
m[2, 5] = \min \left\{ m[2, 3] + m[4, 5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125, \\
          m[2, 4] + m[5, 5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11375 \right. \\
     = 7125 .
\]
Longest common subsequence

Let \( X = x_1, x_2, \ldots x_n \) and \( Y = y_1, y_2, \ldots y_m \) be sequences of lengths \( n \) and \( m \). We want to find a longest common subsequence of \( X \) and \( Y \).

Let \( L[i, j] \) denote the optimal answer for \( x_1, x_2, \ldots x_i \) and \( y_1 y_2, \ldots y_j \). Note that we need to compute \( L[n, m] \).

Define \( L[i, j] = 0 \), if \( i = 0 \), or \( j = 0 \).

\[
L[i, j] = \begin{cases} 
0, & \text{if } i = 0 \text{ or } j = 0 \\
L[i-1, j-1] + 1, & \text{if } x_i = y_j \\
\max\{L[i, j-1], L[i-1, j]\}, & \text{if } x_i \neq y_j
\end{cases}
\]
Longest common subsequence

Algorithm (outline)
We use a 2 dimensional array to store L which has \( n + 1 \) rows and \( m + 1 \) columns. We set the first row and the first column of L to 0. We use the recurrence relation to compute the entries of L in a major row order, that is, first row 0, then, row 1, etc. Then, computing \( L[i, j] \) in row i will take \( O(1) \) time. So time complexity of computing \( L[n, m] \) is \( O(nm) \).
A 2 times optimal approximation algorithm for triangular TSP

Let $G = (V, E)$ be a complete graph with weight function $w : E \rightarrow \mathbb{R}^+$. The Traveling salesman Problem (TSP) is to find a Hamiltonian cycle with minimum weight. An instance of TSP is called triangular if $w(x, y) + w(y, z) \geq w(x, z)$, for all $x, y, z \in V$.

**Lemma**

Let $O$ be an optimal tour for TSP and let $T$ be a minimum spanning tree. Then $w(O) \geq w(T)$.

**Proof.** Note that $O - \{e\}$ is a spanning path of $G$. Thus $w(O - \{e\}) \geq w(T)$, and consequently $w(O) \geq w(T)$. □.
A 2 times optimal approximation algorithm for triangular TSP

Lemma

Let $T$ be a minimum spanning tree. Pick a root $r$ and let $L$ be a list of edges in $T$ that is obtained by pre-order traversal of $T$. Then $w(L) = 2w(T)$. □

Theorem

Let $G = (V, E)$ and let $w$ be a weight function that satisfies triangular inequality. Let $T$ be a minimum spanning tree of $G$. Pick a root $r$ and let $L$ be a list of edges in $T$ that is obtained by pre-order traversal of $T$. Let $C$ be a Hamiltonian cycle that is obtained by short circuiting $L$. Then

$$2w(O) \geq 2w(T) \geq w(C) \geq w(O) \geq w(T)$$

Thus $C$ is a 2 times optimal approximate solution for triangular TSP. □
A 2 times optimal approximation algorithm for triangular TSP

Algorithm (outline)
First, compute a minimum spanning tree $T$, in $O(n^2)$, using Prim’s algorithm. Now select a root $r$ and perform a pre-order traversal to obtain a walk $L$, and let $C$ be the Hamiltonian cycle which is obtained by short circuiting $L$. Then, it follows from last Theorem, that, $C$ is a 2 times optimal approximate solution for triangular TSP.
Figure 35.2 The operation of APPROX-TSP-TOUR. (a) A complete undirected graph. Vertices lie on intersections of integer grid lines. For example, $f$ is one unit to the right and two units up from $h$. The cost function between two points is the ordinary euclidean distance. (b) A minimum spanning tree $T$ of the complete graph, as computed by MST-PRIM. Vertex $a$ is the root vertex. Only edges in the minimum spanning tree are shown. The vertices happen to be labeled in such a way that they are added to the main tree by MST-PRIM in alphabetical order. (c) A walk of $T$, starting at $a$. A full walk of the tree visits the vertices in the order $a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$. A preorder walk of $T$ lists a vertex just when it is first encountered, as indicated by the dot next to each vertex, yielding the ordering $a, b, c, h, d, e, f, g$. (d) A tour obtained by visiting the vertices in the order given by the preorder walk, which is the tour $H$ returned by APPROX-TSP-TOUR. Its total cost is approximately 19.074. (e) An optimal tour $H^*$ for the original complete graph. Its total cost is approximately 14.715.
Approximating Vertex Cover

Let \( G = (V, E) \) be a graph, and let \( S \subseteq V \). \( S \) is a vertex cover, if any edge in \( E \) has at least one endpoint in \( S \). A vertex cover is a minimum vertex cover, if it contains the smallest number of vertices.

Let \( G = (V, E) \) be a graph and let \( M \subseteq E \). \( M \) is called a matching if no two edges in \( M \) have an end point in common. A matching \( M \) is a maximum matching if it has contained the largest number of edges. A matching \( M \) is maximal if for any \( e \in E - M \), \( M \cup \{e\} \) is not a matching.

- Note that a maximal matching may not be a maximum matching. Also, note that a simple greedy algorithm computes a maximal matching in \( O(|V| + |E|) \) time.
Theorem

Let $C$ be a vertex cover, and $M$ be a matching in $G$. Then $|C| \geq |M|$.

Theorem

Let $M$ be a maximal matching in $G$, and let $C$ be the set of all endpoints for edges in $M$. (thus $|C| = 2|M|$.) Then $C$ is a vertex cover in $G$. 

Approximating Vertex Cover

A 2 Times Optimal Approximation Algorithm

- Compute a maximal matching $M$ in $O(|V| + |E|)$ time, and let $C$ be the set of all $2|M|$ end points of edges in $M$.

By the second theorem, $C$ is a vertex cover with $|C| = 2|M|$. Now let $C^*$ be a minimum vertex cover, by the first theorem $|C^*| \geq |M|$. Thus, $|C| = 2|M| \leq 2|C^*|$. 
For the definitions see the CLR book.

**Theorem**

*Any comparison sort Algorithm requires $\Omega(n \log(n))$ time (comparisons) in the worst case.*

**Proof.** Use the decision tree model. Assume that the height is $h$. Since the tree is binary and must have $n!$ leaves, we should get $2^h \geq n!$ implying $h \geq \log(n!)$. But $\log(n!) = \Theta(n \log(n))$ (Why ?) so the claim follows. $\square$