Dynamic Programming  Dynamic programming tackles the situations where a problem can be subdivided, but the number of subproblems is not a constant.

It achieves efficiency by organizing the computation in such a way that a subproblem never gets solved more than once.

This is achieved through a tableau method: the computation proceeds from small subproblems to large subproblems, once a subproblem is solved its solution is stored (as if in a table), the solution to a larger subproblem is then computed from the solutions to smaller subproblems which are already in the table.
Matrix chain product

$m_{ij}$: minimum cost for multiplying the chain $M_i, \ldots, M_j$.

Observe

1. In whatever order we perform the chain product, the last matrix product is $A \times B$ where $A = M_i \times \ldots \times M_k$, and $B = M_{k+1} \times \ldots \times M_j$, for some $k$, $i \leq k < j$.

2. Minimum cost for obtaining $A$ through chain product is $m_{i,k}$

3. Minimum cost for obtaining $B$ through chain product is $m_{k+1,j}$

Assuming that the last product is $A \times B$, then the minimum cost for the whole chain $M_i$ through
$M_j$ is (cost for mult. $A$ with $B$) + $m_{i,k}$ + $m_{k+1,j}$.

Hence for $i < j$:

$$m_{ij} = \min_{i \leq k < j} r_{i-1}r_kr_j + m_{i,k} + m_{k+1,j}.$$ 

Index of problem: $j - i$. The smaller the chain, the smaller the subproblem.
Longest Common Subsequence (LCS) Problem

A subsequence of a sequence $S = <s_1, ..., s_n>$ is of the form $<s_{k_1}, ..., s_{k_m}>$ where $1 \leq k_1 < k_2 ... < k_m \leq n$.

$LCS$ problem: Given two sequences $X$ and $Y$, to find a common subsequence of $X$ and $Y$ of longest possible length.
Let \( X = \langle x_1, ..., x_m \rangle, Y = \langle y_1, ..., y_n \rangle \).

Suppose \( Z = \langle z_1, ..., z_k \rangle \) is a l.c.s. of \( X \) and \( Y \).

A case analysis reveals a suboptimality structure of the problem and leads to a dynamic programming solution.

Case (1): \( x_m = y_n \) (i.e. \( X \) and \( Y \) ends in the same char.)

Then that char., say \( c \), must be the last char of \( Z \).

Else \( \langle z_1, ..., z_k \rangle \) is a c.s. of \( X_{m-1} \) and \( Y_{n-1} \), but then \( \langle Z, c \rangle \) is a longer c.s., a contradiction.

Hence \( \langle z_1, ..., z_{k-1} \rangle \) is a c.s. of \( X_{m-1} \) and \( Y_{n-1} \).

Moreover \( \langle z_1, ..., z_{k-1} \rangle \) is a l.c.s. of \( X_{m-1} \) and \( Y_{n-1} \).
Case (2): $x_m \neq y_n$.

Then either $z_k \neq x_m$ or $z_k \neq y_n$.

- if $z_k \neq x_m$ then $Z$ is a l.c.s. of $X_{m-1}$ and $Y$.

- if $z_k \neq y_n$ then $Z$ is a l.c.s. of $X$ and $Y_{n-1}$. 
Optimal substructure: l.c.s. of two seq. contains l.c.s. of two prefix subseq.

Subproblems: For all prefix subseq $X_i$ and $Y_j$, $i \leq m$, $j \leq n$, find an l.c.s. of $X_i$ and $Y_j$.

Let $c(i, j) =$ length of the l.c.s. of $X_i$ and $Y_j$.

Then $c(i, j)$

$$c(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
(c(i-1, j-1) + 1 & \text{if } i, j > 0, x_i = y_j \\
\max\{c(i-1, j), c(i, j-1)\} & \text{if } i, j > 0, x_i \neq y_j 
\end{cases}$$

Time: $O(mn)$. 
Optimal binary search trees

Given a seq. of keys $< k_1, ..., k_n >$ where $k_1 < ... < k_n$ with prob. $p_i$ that a search is for $k_i$, $i = 1, ..., n$ and prob. $q_i$ that a search is for a key $x$ in the range $k_i < x < k_{i+1}$ for $i = 0, ..., n$ with $k_0 = -\infty$ and $k_{n+1} = +\infty$.

To construct a binary search tree $T$ on $K$ so that

$$W(T) = \sum_{i=1}^{n} p_i (d_T(k_i) + 1) + \sum_{i=0}^{n} q_i (d_T(d_i) + 1)$$

is minimized.

Here $d_i$ corresponds to the range $k_i < x < k_{i+1}$.
Observe that if $k_r$ is the root of a search tree, then

the left subtree $T_1$ is a search tree for $<k_1, \ldots, k_{r-1} >$

the right subtree $T_2$ is a search tree for $< k_{r+1}, \ldots, k_n >$

$d_{T_1}(x) = d_T(x) - 1$ for $x$ on $T_1$, $d_{T_2}(x) = d_T(x) - 1$ for $x$ on $T_2$, hence

$W(T) = W(T_1) + w(T_2) + \sum_i p_i + \sum_i q_i$

So $T$ optimal for $< k_1, \ldots, k_n > \Rightarrow T_i$ optimal (optimal substructure)
More generally, let

\[ e(i, j) := \text{wt of min b.s.t. on } k_i, \ldots, k_j \text{ and} \]

\[ w(i, j) = \sum_{x=i}^{j} p_x + \sum_{x=i-1}^{j} q_x. \]

Then

\[ e(i, j) = \min_{i \leq r \leq j} e(i, r-1) + e(r+1, j) + w(i, j), i \leq j \]

\[ e(i, i-1) = q_{i-1} \text{ (b.s.t. for search ending up in} \]

\[ k_{i-1} < x < k_i) \]