Dynamic Programming:

Dynamic programming has similarities to both divide-and-conquer and backtracking. It is similar to divide and conquer in that sub-problems are generated. However, it also shares characteristics with back-tracking algorithms in the sense that we often do not know exactly which sub-problems definitively must be solved; i.e., we are stuck with solving "all of them".

Dynamic programming uses a table to avoid redundant computation of sub-problems. Problem characteristics where dynamic programming is applicable include:

1. The problem can be solved by a brute-force approach involving a set of sub-problems.
2. The sub-problems overlap.
3. The sub-problems can be organized in a table, and indexed in a straightforward manner.

Solutions to sub-problems are placed in a table as they become known. There are two variations to managing the table.

"On demand" style

"On demand" style algorithms use a recursive divide-and conquer approach, but before making a recursive call to solve a sub-problem, the table is checked to see if the solution to that particular sub-problem is already known. If a solution exists in the table, then the value from the table is used, and no recursive call is made. If the solution to a sub-problem is not in the table (i.e., not known), then the recursive call is made, and the solution found by the recursive call is immediately placed in the table on return from the recursive call. The solution to future instances of the same sub-problem will be found in the table.

"Pre-ordered" style

It is usually possible to re-order the computation such that the solution each sub-problem is placed in the table in advance of needing that solution. We illustrate this style of dynamic programming with the following problem:

Optimal Order for Matrix Multiplication

Let

$$\hat{M} = M_1 \times M_2 \times M_3 \times ... \times M_n \tag{1}$$

be a matrix product of interest,[1] and let $r_0, r_1, r_2, ..., r_n$ be an array of integers representing the dimensions of the matrices.

I.e., matrix M_j is size r_{j-1} rows by r_j columns.

Observe that the cost of multiplying $M_j \times M_{j+1}$ is the scalar product: $r_{j-1} r_j r_{j+1}$.

[1] In an example shown later, we will see that the order in which the matrix products are performed can make a large difference in the number of scalar multiply operations which must be performed.

Define a table with entries $m_{i,j}$, where

$$m_{i,j} = \text{minimal cost of multiplying } M_i \times M_{i+1} \times ... \times M_j$$

The key observation to solving this problem is to observe:

$$m_{i,j} = \begin{cases} 0 \text{ if } i = j \\ \min_{i \le k < j} \quad m_{i,k} + m_{k+1,j} + r_{i-1} r_k r_j \quad \text{otherwise} \end{cases} \tag{2}$$

To better understand equation (2), observe that $m_{i,k}$ is the optimal cost of multiplying matrices M_i through M_k. Similarly, $m_{k+1,j}$ is the optimal cost of multiplying matrices M_{k+1} through M_j. We claim that by taking the minimum over all possible split points k, solving the sub-problems optimally, and including the cost of the matrix multiplication at point k, we find the optimal cost of multiplying M_i through M_j.

Notice that the dimensions of matrix product $M_i \times M_{i+1} \times ... \times M_k$ are r_{i-1} by r_k. Similarly, the dimensions of the matrix product $M_{k+1} \times M_{k+2} \times ... \times M_j$ are r_k by r_j. Therefore the cost of combining results of the two sub-products is $r_{i-1} r_k r_j$.

Here's the clever part of the dynamic programming: If we compute the numbers $m_{i,j}$ in the order of increasing difference between the subscripts (i.e., $j - i$), the values that we need in the right hand side of equation will already be in the table at the time we need them.

Algorithm

Input: $r_0, r_1, r_2, ...r_n$

Output: The optimal cost of multiplying matrices with the given dimensions.

Method:

for $i = 1$ to n do
    $m_{i,i} = 0$ // One matrix, therefore zero multiplies.
for $\ell = 1$ to $n - 1$ do {
    for $i = 1$ to $n - \ell$ do {
        $j = \ell + i$
        $m_{i,j} = \min_{i \le k < j} \quad m_{i,k} + m_{k+1,j} + r_{i-1} r_k r_j$
    } // end for i
} // end for l

output $m_{1,n}$

Example

Suppose we wish to compute:

$$M = M_1 \times M_2 \times M_3 \times M_4$$

Also suppose the matrix sizes are:

Matrix	M_1	M_2	M_3	M_4
Size	10×20	20×50	50×1	1×100

Computing:

$$M = M_1 \times (M_2 \times (M_3 \times M_4)) \Rightarrow 125000 \text{ operations}$$

Computing:

$$M = (M_1 \times (M_2 \times M_3)) \times M_4 \quad \Rightarrow 2200 \text{ operations}$$

Sample Dynamic Programming Computation

$m_{11} = 0$	$m_{22} = 0$	$m_{33} = 0$	$m_{44} = 0$
$m_{12} = 10000$	$m_{23} = 1000$	$m_{34} = 5000$	
$m_{13} = 1200$	$m_{24} = 3000$		
$m_{14} = 2200$			

Relation to Catalan Numbers

The number of ways to parenthesize the product of n matrices in equation (1) is given by the *Catalan number* C_{n-1}. The Catalan numbers are defined by the equations:

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^{n} C_i \, C_{n-i} \quad \text{for } n \geq 0;$$

A closed-form expression for the Catalan numbers is given by:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} = \prod_{k=2}^{n} \frac{n+k}{k} \qquad \text{for } n \geq 0.$$

The first few Catalan numbers for $n = 0, 1, 2, 3, ...$ are

$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, ..$$

Asymptotically, the Catalan numbers are approximately exponential. I.e.:

$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$