Dynamic Programming

→ can often be used to take problems which seem to require exponential time and produce polynomial-time algorithms to solve them

→ Matrix Chain-Product

Suppose we are given a collection of n 2-D matrices for which we want to compute

$$A = A_0 \cdot A_1 \cdot A_2 \cdots A_{n-1},$$

where A_i is a $d_i \times d_{i+1}$ matrix, for $i = 0, 1, 2, \ldots, n-1$.

In the standard matrix mult. algorithm:

$$A[i][j] = \sum_{k=0}^{e-1} B[i][k] \cdot C[k][j]$$

This implies that matrix multiplication is associative -- $B \cdot (C \cdot D) = (B \cdot C) \cdot D$

→ Defining Subproblems

→ we can significantly improve performance of brute-force algorithm

→ matrix-chain-product problem can be split into subproblems

- → in this case, each subproblem is used to compute the best parenthesization for some expression $A_i \cdot A_{i+1} \cdots A_j$

→ Characterizing Optimal Solutions

→ it is possible to characterize an optimal solution to a particular subproblem in terms of optimal solutions to its subproblems

→ we can compute $N_{i,j}$ by considering each place k where we could put the final multiplication and taking the minimum over all such choices

→ Designing Dynamic Programming Algo

we can characterize the optimal solution as:

$$N_{i,j} = \min_{i \le k < j} \{ N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1} \}$$

where $N_{i,i} = 0$ since no work is needed for a single matrix.

$N_{i,j}$ is the minimum taken over all possible places to perform the final multiplication of the # of multiplications needed to compute each subexpr. plus the number of multiplications needed to perform the final matrix multiplication.