Dynamic Programming

= Divide-and-conquer optimization

with memoization.

make a set of choices so that the result optimizes some scoring function

augment a function with an input/output table remembering output values as they are generated to avoid recomputation.

- we design a recursive function that computes the optimal achievable score (and memoizes that score)

example Matrix Chain Multiplication Parenthesization

A matrix has 2 parameters (the number of rows p, the number of columns q)

A $p \times q$ matrix

matrix multiply

$M_1 \cdot M_2$ is computed using pqr scalar multiplies

$p \times q \quad q \times r$ and is $p \times r$

matrix multiply is associative.

$$(M_1 \cdot M_2) \cdot M_3 = M_1 \cdot (M_2 \cdot M_3)$$

$15 \times 1 \quad 1 \times 1 \quad 1 \times 1$ | $15 \times 1 \quad 1 \times 1 \quad 1 \times 1$

cost 5: $M_{12} \cdot M_3$ (15×1)

cost 15: M_{12}

cost 15

30

cost 1: $M_1 \cdot M_{23}$

cost 15

16

M_{123}

Problem:

Given $M_1 \cdot M_2 \cdots M_n$

insert parentheses to multiply at cheapest cost measured in scalar multiplies

Brute force approach: consider each parenthesization to find one of best score.

How many parenthesizations are there? $(M_1 \cdot M_2 \cdot M_3) \cdot (M_4 \cdots M_n)$

$T(n)$ = # of parenthesizations possible for n matrices

$$= \sum_{i=1}^{n-1} \text{ways to parenthesize with last multiply just after } M_i$$

$$T(n) = \sum_{i=1}^{n-1} T(i) \cdot T(n-i) \qquad T(1) = 1$$

$\Rightarrow$ can show $T(n)$ as specified is exponential using induction e.g. $T(n) \geq 2^{n-1}$

idea: try each possible last multiply position and compute (using recursion) its best score achievable if we commit to having the last multiply here.

$M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdots M_n$

$p_0 \times p_1 \quad p_1 \times p_2 \quad p_2 \times p_3 \quad p_3 \times p_4 \quad p_{n-1} \times p_n$

optscore $(M_1 \cdots M_n)$
i.e. optscore $(1,n)$

— e.g. commit to $(M_1 \cdots M_i)$ | $(M_{i+1} \cdots M_n)$

$p_0 \times p_i$ | $p_i \times p_n$

and compute

$$\text{optscore}(1,n) = \min_{i=1}^{n-1} \left[\text{optscore}(1,i) + \text{optscore}(i+1,n) + p_0 p_i p_n \right]$$

but record best choice $[1,n] = \operatorname{argmin}$

$\text{optscore}(i,i) = 0$ for each i

allows reconstructing solution choices

can memoize with table having $< n^2$ entries.
each entry has linear cost to compute (excluding recursion)

n^3 algorithm

Using best-choice array to reconstruct the parenthesization.

```
print_solution(i,j)   // print a parenthesization of M_i ... M_j
if (i==j)
    print(" "M_i" ");
else {   print("(");
         print_solution(i, best_choice[i,j]);
         print(".");
         print_solution(best_choice[i,j]+1, j);
         print(")");
     }
```