Dynamic Programming

- Algorithm design technique, usually for optimization and counting problems.
- Useful when solution can be expressed **recursively**, but the **same subproblems** appear in **different recursive calls**.
- Strategy: **solve** subproblems and **store** results. Large problems are still decomposed into subproblems, but solutions to subproblems are **"looked up"**.

- Typical Approach
-Find recursive description of optimal cost.
- Tabulate subproblems of subproblems
- Tabulate how opt. soln. was generated
-construct solution.

Example: Fibonacci Numbers

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

Compute F_n?

Recursive strategy:

F(n)
 if $n = 0$ **then return** 0
 else if $n = 1$ **then return** 1
 else return F($n - 1$)+F($n - 2$)

Computation tree for F(5):

Dynamic Programming Strategy

F(n)

▷ uses local $A[1..n]$

$A[0] := 0; \quad A[1] := 1;$

for $i = 2$ **to** n **do**

$A[i] := A[i-1] + A[i-2]$

return $A[n]$

Recursive strategy takes time

$\geq$ number of nodes in tree

$\geq F_n \in \Omega((1.6)^n)$.

Dynamic programming strategy takes time

$$\Theta(n)$$

Multiplying a Sequence of Matrices

$$\begin{array}{ccccc} \boldsymbol{C} & \leftarrow & \boldsymbol{A} & \cdot & \boldsymbol{B} \\ \boldsymbol{m \times p} & & \boldsymbol{m \times n} & & \boldsymbol{n \times p} \end{array}$$

requires $\boldsymbol{mnp}$ scalar mults. by usual alg.

Multiplication of matrices is **associative**:

$$\boldsymbol{(AB)C = A(BC)}$$

However, # of scalar mults. may be different.

Example - **Multiply:** $\boldsymbol{ABC}$

$$\begin{array}{ccc} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{5 \times 100} & \boldsymbol{100 \times 2} & \boldsymbol{2 \times 1} \end{array}$$

$\boldsymbol{(AB)C}$ vs. $\boldsymbol{A(BC)}$?

In computing product of a sequence of matrices, how to associate to **minimize the number of scalar multiplications**?

Need product:

$$A_1 \cdot A_2 \cdot A_3 \cdot \cdots \cdot A_n$$

where A_i is $d_{i-1} \times d_i$

Given: $d_0, d_1, \ldots, d_n$

Goal: minimize number of scalar multiplications

Think top-down. Which is best?

$$(A_1)(A_2 \cdots A_n)$$

$$(A_1 A_2)(A_3 \cdots A_n)$$

$$(A_1 \cdots A_3)(A_4 \cdots A_n)$$

$$\vdots$$

$$(A_1 \cdots A_{n-1})(A_n)$$

$M(i,j) =$ min. # of scalar mults. to compute:

$$A_i \cdot A_{i+1} \cdot \cdots \cdot A_j$$

Then the min. # of mults. if split at k:

$$(A_i \cdots A_k)\ (A_{k+1} \cdots A_j)$$

is:

$$M(i,k) + M(k+1,j) + d_{i-1}d_k d_j \qquad (*)$$

So, for $i < j$:

$M(i,j)$ is the minimum of (*)

over all k: $i \leq k \leq j-1$

For $i = j$:

$M(i,i) = 0$

Now have recurrence ...

... but don't compute recursively!

Dynamic programming to compute $M(1, n)$

for $i \leftarrow 1$ **to** n **do**

$\quad M[i, i] \leftarrow 0$

for $i \leftarrow n - 1$ **downto** 1 **do**

$\quad$ **for** $j \leftarrow i + 1$ **to** n **do**

$$M[i, j] \leftarrow \min_{i \le k \le j-1} \{M[i, k] + M[k + 1, j] + d_{i-1} d_k d_j\}$$

Total time: $\Theta(n^3)$

In computing $M[i, j]$, save in $S[i, j]$ the value of k where the minimum is achieved:

$$
\begin{array}{ccccccc}
 & j & \longrightarrow & & & & \\
i & / & 1 & 1 & 3 & 3 & 3 \\
\downarrow & / & / & 2 & 3 & 3 & 3 \\
 & / & / & / & 3 & 3 & 3 \\
 & / & / & / & / & 4 & 5 \\
 & / & / & / & / & / & 5 \\
 & / & / & / & / & / & /
\end{array}
$$

Memoizing

Initialize table with a special symbol, say $\#$.

Use **recursive** version of program **except** -

when $M[i, j]$ to be called recursively:

- check if $M[i, j] = \#$.
- if not, return value, else
- make recursive call; store result

See text for memoized version of matrix chain multiplication.