# **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

$$egin{aligned} F_0 &= 0 \ F_1 &= 1 \ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

Compute  $F_n$ ?

Recursive strategy:

```
\mathsf{F}(n) if n=0 then return 0 else if n=1 then return 1 else return \mathsf{F}(n-1) + \mathsf{F}(n-2)
```

Computation tree for F(5):

# **Dynamic Programming Strategy**

$$\mathsf{F}(n)$$
  $ightharpoonup \mathsf{uses local}\ A[1..n]$   $A[0] := 0; \quad A[1] := 1;$   $\mathsf{for}\ i = 2\ \mathsf{to}\ n\ \mathsf{do}$   $A[i] := A[i-1] + A[i-2]$   $\mathsf{return}\ A[n]$ 

# Recursive strategy takes time

≥ 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

$$egin{array}{ccccc} C & \leftarrow & A & \cdot & B \ m imes p & m imes n & n imes p \end{array}$$

requires mnp scalar mults. by usual alg.

Multiplication of matrices is **associative**:

$$(AB)C = A(BC)$$

However, # of scalar mults. may be different.

Example - Multiply: ABC

$$(AB)C$$
 vs.  $A(BC)$  ?

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  $d_{i-1} imes d_i$ 

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

Goal: minimize number of scalar multiplications

# Think top-down. Which is best?

$$(A_1)(A_2\cdots A_n)$$
 $(A_1A_2)(A_3\cdots A_n)$ 
 $(A_1\cdots A_3)(A_4\cdots A_n)$ 
 $\vdots$ 
 $(A_1\cdots A_{n-1})(A_n)$ 

$$M(i,j) =$$
 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:

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

So, for i < j:

 $oldsymbol{M}(i,j)$  is the minimum of (\*)

over all k: i < k < j-1

For i = j:

$$M(i,i)=0$$

Now have recurrence ...

... but don't compute recursively!

# Dynamic programming to compute M(1,n)

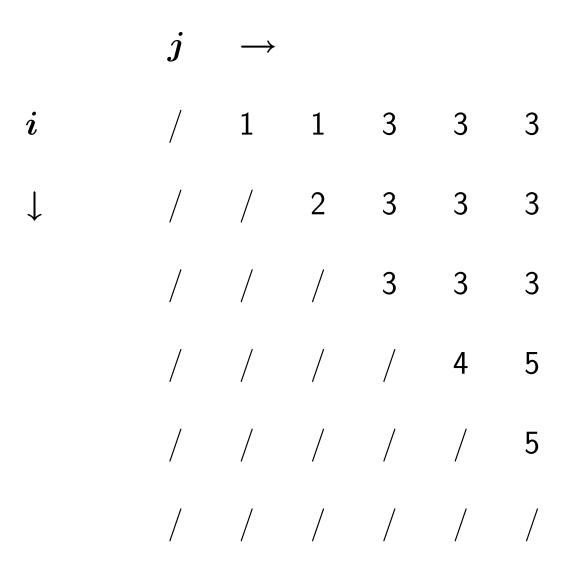
for 
$$i \leftarrow 1$$
 to  $n$  do $M[i,i] \leftarrow 0$ 

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

for 
$$j \leftarrow i+1$$
 to  $n$  do $M[i,j] \leftarrow \min_{i \leq k \leq j-1} \{M[i,k] + M[k+1,j] + d_{i-1}d_kd_j\}$ 

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

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



# Memoizing

Initialize table with a special symbol, say #.

Use  $\mathbf{recursive}$  version of program  $\mathbf{except}$  - when M[i,j] to be called recursively:

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

See text for memoized version of matrix chain multiplication.