Analysis of Algorithms
CS 477/677

Instructor: Monica Nicolescu

Lecture 16

Dynamic Programming

- An algorithm design technique used for **optimization problems**
  - Find a solution with the **optimal value** (minimum or maximum)
  - A set of **choices** must be made to get an optimal solution
  - There may be multiple solutions that return the optimal value: we want to find one of them

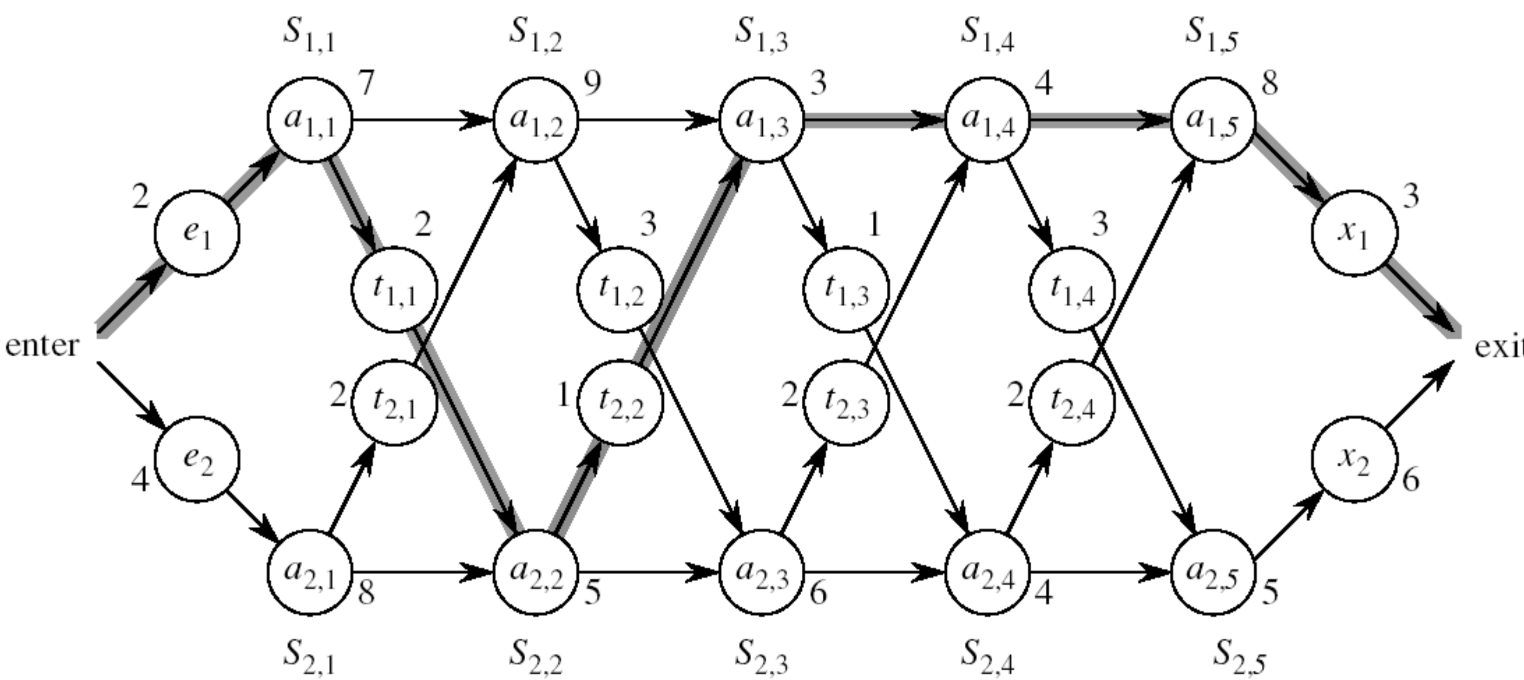
Dynamic Programming Algorithm

1. Characterize the structure of an optimal solution
   - **Top down:** how can an optimal value for a problem be obtained from combinations of optimal solutions to similar, smaller problems of the same type
2. Recursively define the value of an optimal solution
   - **Top down:** write a recursive formula based on the step above
3. Compute the value of an optimal solution
   - **Bottom up:** compute ***"smaller subproblems"*** first, store **values** and **choices** made at each step
4. Construct an optimal solution
   - **Top down:** start with last choice made and backtrack, finding all choices made

Assembly Line Scheduling

- Problem:

  What stations should be chosen from line 1 and what from line 2 in order to minimize the total time through the factory for one car?



Dynamic Programming Algorithm

1. Characterize the structure of an optimal solution
   – Fastest time through a station depends on the fastest time on previous stations
2. Recursively define the value of an optimal solution
   – $f_1[j] = \min(f_1[j - 1] + a_{1,j}, f_2[j - 1] + t_{2,j-1} + a_{1,j})$
3. Compute the value of an optimal solution in a bottom-up fashion
   – Fill in the fastest time table in increasing order of j (station #)
4. Construct an optimal solution from computed information
   – Use an additional table to help reconstruct the optimal solution

Matrix-Chain Multiplication

Problem: given a sequence $\langle A_1, A_2, \ldots, A_n \rangle$ of matrices, compute the product:

$$A_1 \bullet A_2 \bullet\bullet\bullet A_n$$

- Matrix compatibility:

$$C = A \bullet B$$

$$col_A = row_B$$

$$row_C = row_A$$

$$col_C = col_B$$

$$A_1 \bullet A_2 \bullet A_i \bullet A_{i+1} \bullet\bullet\bullet A_n$$

$$col_i = row_{i+1}$$

Matrix-Chain Multiplication

- In what order should we multiply the matrices?

$$A_1 \bullet A_2 \bullet\bullet\bullet A_n$$

- Matrix multiplication is associative:
- E.g.: $A_1 \bullet A_2 \bullet A_3 = ((A_1 \bullet A_2) \bullet A_3)$

$$= (A_1 \bullet (A_2 \bullet A_3))$$

- Which one of these orderings should we choose?
  - The order in which we multiply the matrices has a significant impact on the overall cost of executing the entire chain of multiplications

MATRIX-MULTIPLY(A, B)

if columns[A] ≠ rows[B]

 then error “incompatible dimensions”

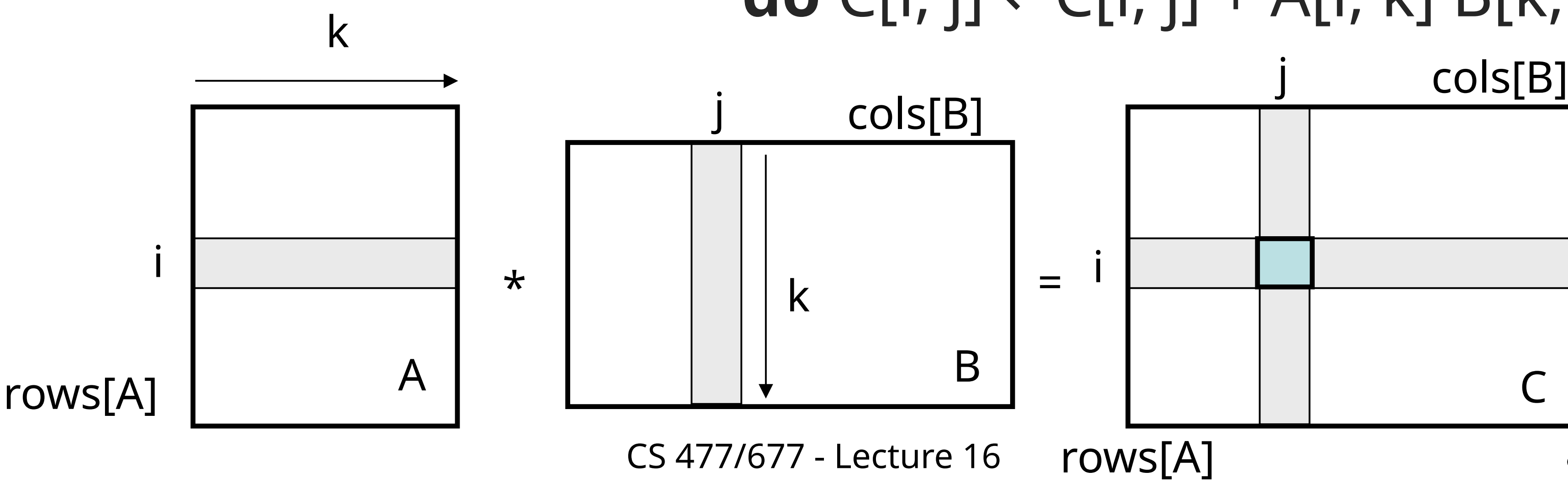
 else for i ← 1 to rows[A]

    do for j ← 1 to columns[B]

      do C[i, j] = 0

       for k ← 1 to columns[A]

        do C[i, j] ←C[i, j] + $\overbrace{A[i, k]\ B[k, j]}^{\text{rows[A] • cols[A] • cols[B] multiplications}}$



Example

$$A_1 \bullet A_2 \bullet A_3$$

- A_1: 10 x 100
- A_2: 100 x 5
- A_3: 5 x 50

1. $((A_1 \bullet A_2) \bullet A_3)$: $A_1 \bullet A_2$ takes 10 x 100 x 5 = 5,000

   (its size is 10 x 5)

   $((A_1 \bullet A_2) \bullet A_3)$ takes 10 x 5 x 50 = 2,500

   Total: 7,500 scalar multiplications

2. $(A_1 \bullet (A_2 \bullet A_3))$: $A_2 \bullet A_3$ takes 100 x 5 x 50 = 25,000

   (its size is 100 x 50)

   $(A_1 \bullet (A_2 \bullet A_3))$ takes 10 x 100 x 50 = 50,000

   Total: 75,000 scalar multiplications

one order of magnitude difference!!

Matrix-Chain Multiplication

- Given a chain of matrices $\langle A_1, A_2, \ldots, A_n \rangle$, where for i = 1, 2, …, n matrix A_i has dimensions p_{i-1}x p_i, fully parenthesize the product $A_1 \bullet A_2 \bullet \bullet \bullet A_n$ in a way that minimizes the number of scalar multiplications.

$$A_1 \quad \bullet \quad A_2 \quad \bullet \bullet \bullet \; A_i \quad \bullet \; A_{i+1} \quad \bullet \bullet \bullet \; A_n$$

$$p_0 \times p_1 \quad p_1 \times p_2 \qquad p_{i-1} \times p_i \quad p_i \times p_{i+1} \qquad p_{n-1} \times p_n$$

1. The Structure of an Optimal Parenthesization

- Notation:

$$A_{i\ldots j} = A_i\, A_{i+1} \bullet\bullet\bullet A_j,\ i \le j$$

- For i < j:

$$\begin{aligned} A_{i\ldots j} &= A_i\, A_{i+1} \bullet\bullet\bullet A_j \\ &= A_i\, A_{i+1} \bullet\bullet\bullet A_k\, A_{k+1} \bullet\bullet\bullet A_j \\ &= A_{i\ldots k}\, A_{k+1\ldots j} \end{aligned}$$

- Suppose that an optimal parenthesization of $A_{i\ldots j}$ splits the product between A_k and A_{k+1}, where $i \le k < j$

Optimal Substructure

$$A_{i...j} = A_{i...k} \, A_{k+1...j}$$

- The parenthesization of the “prefix” $A_{i...k}$ must be an optimal parentesization
- If there were a less costly way to parenthesize $A_{i...k}$, we could substitute that one in the parenthesization of $A_{i...j}$ and produce a parenthesization with a lower cost than the optimum ⇒ contradiction!
- An optimal solution to an instance of the matrix-chain multiplication contains within it optimal solutions to subproblems

2. A Recursive Solution

- Subproblem:

  determine the minimum cost of parenthesizing

  $A_{i\ldots j} = A_i A_{i+1} \bullet\bullet\bullet A_j$ for $1 \le i \le j \le n$

- Let $m[i, j]$ = the **minimum** number of multiplications needed to compute $A_{i\ldots j}$
  - Full problem ($A_{1..n}$): $m[1, n]$
  - $i = j$: $A_{i\ldots i} = A_i \Rightarrow m[i, i] =$ 0, for $i = 1, 2, \ldots, n$

2. A Recursive Solution

Consider the subproblem of parenthesizing

$A_{i\ldots j} = A_i\, A_{i+1} \bullet\bullet\bullet A_j$ for $1 \le i \le j \le n$

$p_{i-1}p_kp_j$

$= A_{i\ldots k}\, A_{k+1\ldots j}$ for $i \le k < j$

$m[i, k]$ $m[k+1,j]$

- Assume that the optimal parenthesization splits the product $A_i\, A_{i+1} \bullet\bullet\bullet A_j$ at k $(i \le k < j)$

$$m[i, j] = \underbrace{m[i, k]}_{\substack{\text{min \# of multiplications}\\ \text{to compute } A_{i\ldots k}}} + \underbrace{m[k+1, j]}_{\substack{\text{min \# of multiplications}\\ \text{to compute } A_{k+1\ldots j}}} + \underbrace{p_{i-1}p_kp_j}_{\substack{\text{\# of multiplications}\\ \text{to compute } A_{i\ldots k}A_{k\ldots j}}}$$

2. A Recursive Solution

$$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$$

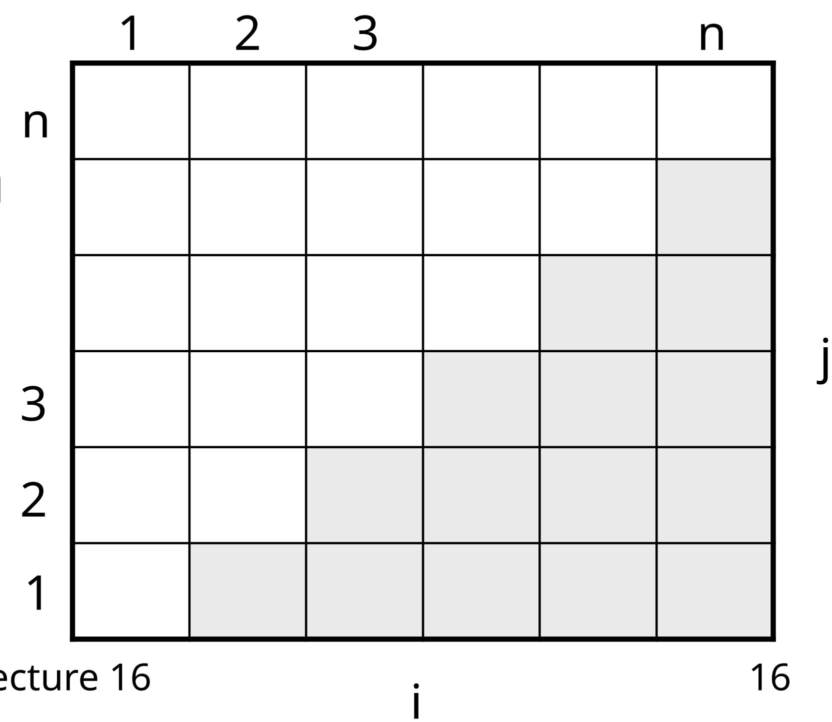
- We do not know the value of k
  - There are j – i possible values for k: k = i, i+1, ..., j-1
- Minimizing the cost of parenthesizing the product $A_i A_{i+1} \cdots A_j$ becomes:

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\} & \text{if } i < j \end{cases}$$

3. Computing the Optimal Costs

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\} & \text{if } i < j \end{cases}$$

- How many subproblems do we have?
  - Parenthesize $A_{i\ldots j}$ for $1 \le i \le j \le n$ $\Rightarrow \Theta(n^2)$
  - One subproblem for each choice of i and j



3. Computing the Optimal Costs

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\} & \text{if } i < j \end{cases}$$

- How do we fill in table m[1..n, 1..n]?
  - Determine which entries of the table are used in computing m[i, j]

$$A_{i\ldots j} = A_{i\ldots k}\, A_{k+1\ldots j}$$

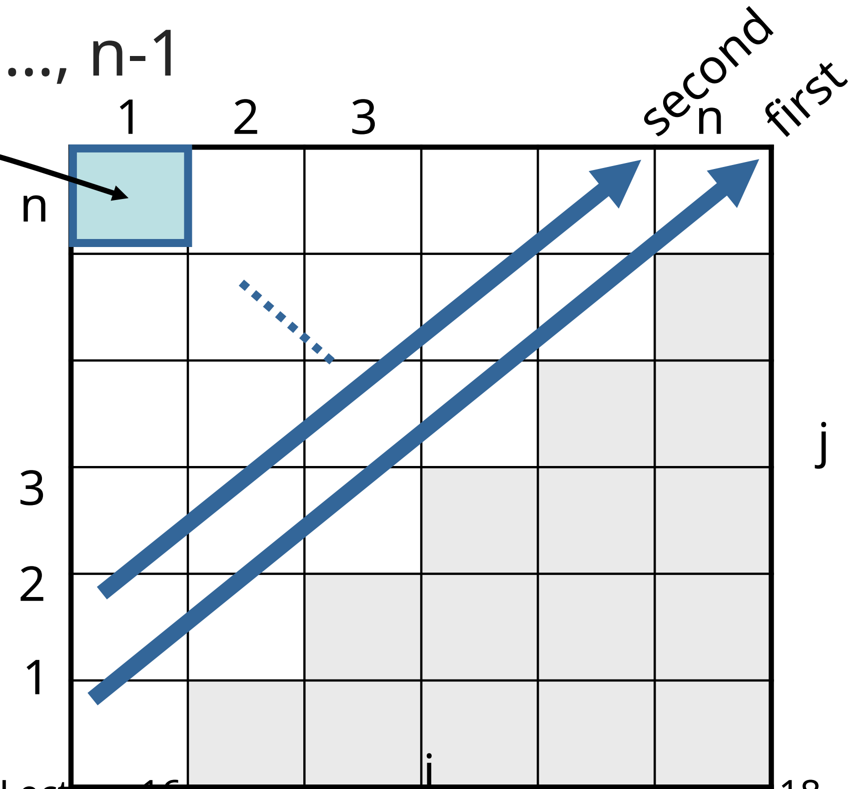
  - Fill in m such that it corresponds to solving problems of increasing length

3. Computing the Optimal Costs

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\} & \text{if } i < j \end{cases}$$

- Length = 1: i = j, i = 1, 2, ..., n
- Length = 2: j = i + 1, i = 1, 2, ..., n-1

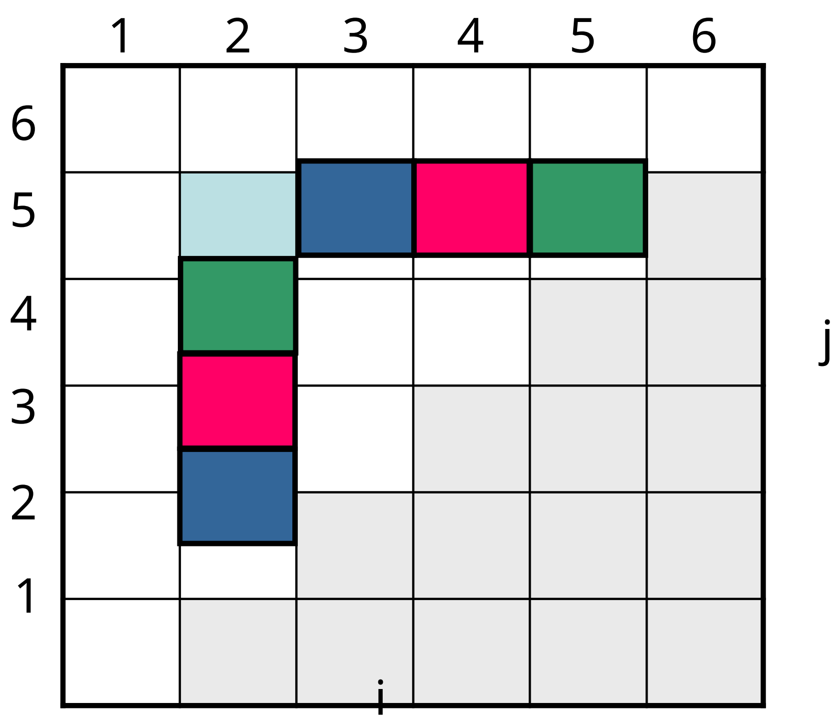
m[1, n] gives the optimal solution to the problem



Compute elements on each diagonal, starting with the longest diagonal.
In a similar matrix s we keep the optimal values of k.

Example: min {m[i, k] + m[k+1, j] + $p_{i-1}p_kp_j$}

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1p_2p_5 & k = 2 \\ m[2,3] + m[4,5] + p_1p_3p_5 & k = 3 \\ m[2,4] + m[5,5] + p_1p_4p_5 & k = 4 \end{cases}$$



- Values m[i, j] depend only on values that have been previously computed

Example $\min \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\}$

	1	2	3
3	2 7500	2 25000	0
2	1 5000	0	
1	0		

Compute $A_1 \bullet A_2 \bullet A_3$

- A_1: 10 x 100 (p_0 x p_1)
- A_2: 100 x 5 (p_1 x p_2)
- A_3: 5 x 50 (p_2 x p_3)

$m[i, i] = 0$ for $i = 1, 2, 3$

$m[1, 2] = m[1, 1] + m[2, 2] + p_0p_1p_2$ (A_1A_2)

$= 0 + 0 + 10 * 100 * 5 = 5{,}000$

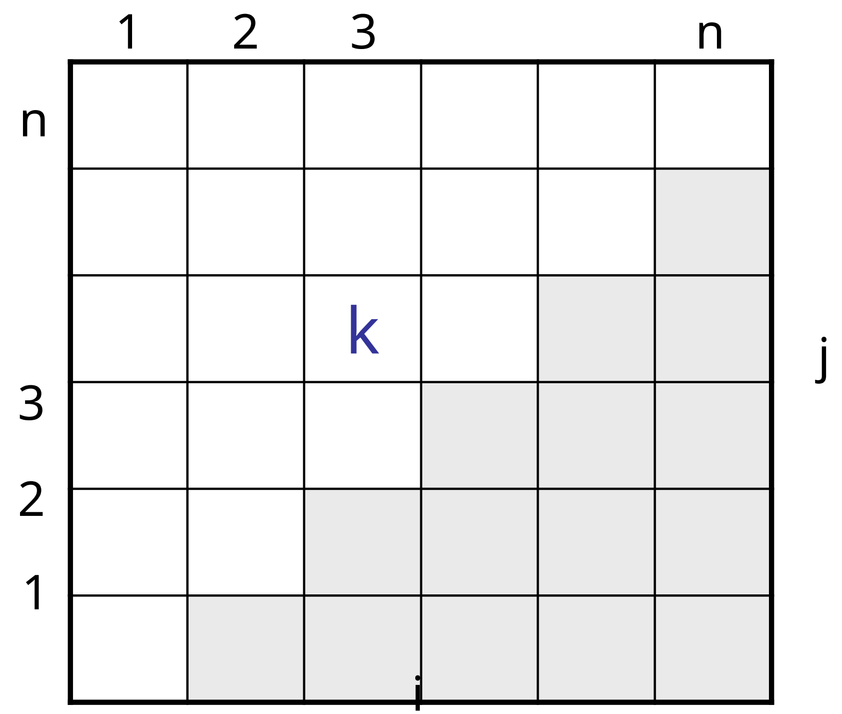
$m[2, 3] = m[2, 2] + m[3, 3] + p_1p_2p_3$ (A_2A_3)

$= 0 + 0 + 100 * 5 * 50 = 25{,}000$

$$m[1, 3] = \min \begin{cases} m[1, 1] + m[2, 3] + p_0p_1p_3 = 75{,}000 & (A_1(A_2A_3)) \\ m[1, 2] + m[3, 3] + p_0p_2p_3 = 7{,}500 & ((A_1A_2)A_3) \end{cases}$$

4. Construct the Optimal Solution

- **Top-down** approach
- Store the optimal choice made at each subproblem
- s[i, j] = a value of k such that an optimal parenthesization of $A_{i..j}$ splits the product between A_k and A_{k+1}

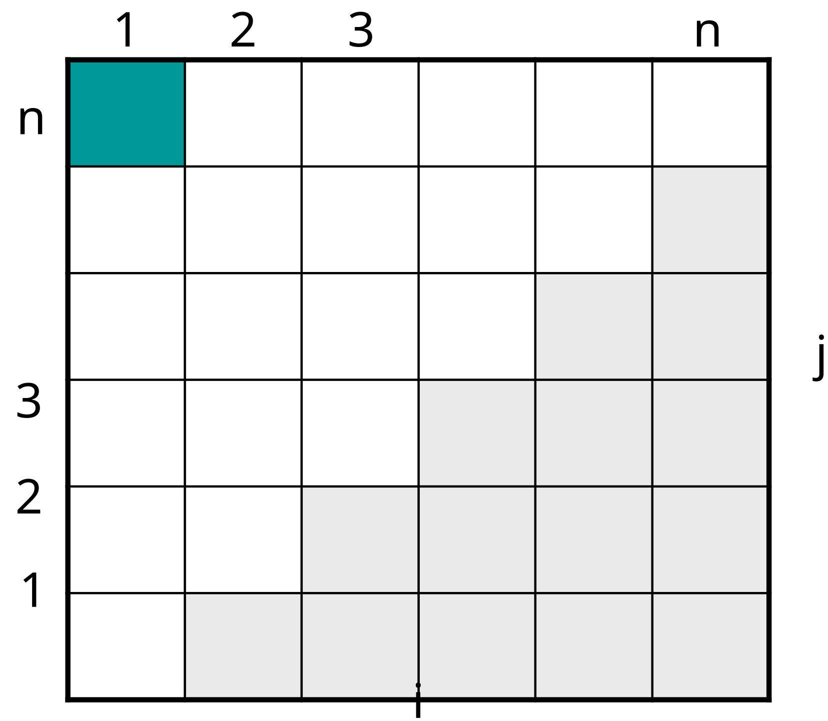


4. Construct the Optimal Solution

- s[1, n] is associated with the entire product $A_{1..n}$
  - The final matrix multiplication will be split at k = s[1, n]

    $A_{1..n} = A_{1..k} \bullet A_{k+1..n}$

    $A_{1..n} = A_{1..s[1,\ n]} \bullet A_{s[1,\ n]+1..n}$
  - For each subproduct recursively find the corresponding value of k that results in an optimal parenthesization



4. Construct the Optimal Solution

- s[i, j] = value of k such that the optimal parenthesization of $A_i\ A_{i+1} \bullet\bullet\bullet A_j$ splits the product between A_k and A_{k+1}

j \ i	1	2	3	4	5	6
6	3	3	3	5	5	-
5	3	3	3	4	-	
4	3	3	3	-		
3	1	2	-			
2	1	-				
1	-					

- s[1, n] = 3 $\Rightarrow A_{1..6} = A_{1..3}\ A_{4..6}$
- s[1, 3] = 1 $\Rightarrow A_{1..3} = A_{1..1}\ A_{2..3}$
- s[4, 6] = 5 $\Rightarrow A_{4..6} = A_{4..5}\ A_{6..6}$

4. Construct the Optimal Solution

```
PRINT-OPT-PARENS(s, i, j)
if i = j
  then print "A"i
  else
       print "("
       PRINT-OPT-PARENS(s, i, s[i, j])
       PRINT-OPT-PARENS(s, s[i, j] + 1, j)
       print ")"
```

	1	2	3	4	5	6
6	3	3	3	5	5	-
5	3	3	3	4	-	
4	3	3	3	-		
3	1	2	-			
2	1	-				
1	-					

j

i

Example: $A_1 \bullet\bullet\bullet A_6$ $((A_1 (A_2 A_3)) ((A_4 A_5) A_6))$

PRINT-OPT-PARENS(s, i, j)

if i = j

  then print "A"$_i$

  else print "("

    PRINT-OPT-PARENS(s, i, s[i, j])

    PRINT-OPT-PARENS(s, s[i, j] + 1, j)

    print ")"

s[1..6, 1..6]

j \ i	1	2	3	4	5	6
6	3	3	3	5	5	-
5	3	3	3	4	-	
4	3	3	3	-		
3	1	2	-			
2	1	-				
1	-					

P-O-P(s, 1, 6)   s[1, 6] = 3

i = 1, j = 6 "("   P-O-P (s, 1, 3) s[1, 3] = 1

i = 1, j = 3 "(" P-O-P(s, 1, 1) ⇒ "A_1"

P-O-P(s, 2, 3) s[2, 3] = 2

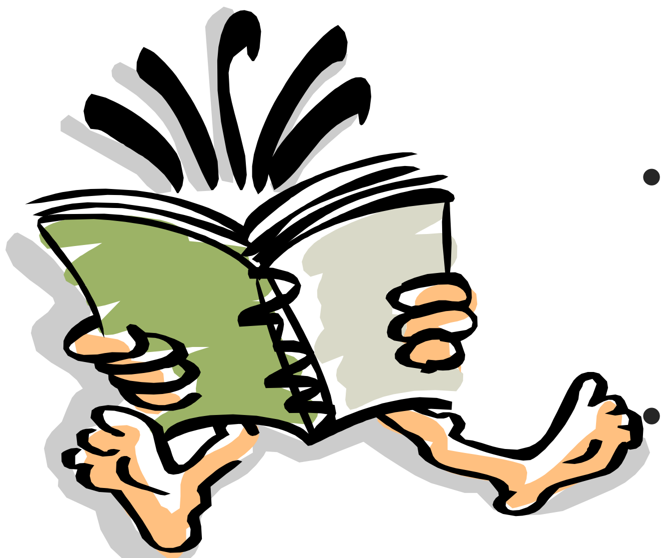
i = 2, j = 3   "(" P-O-P (s, 2, 2) ⇒ "A_2"

P-O-P (s, 3, 3) ⇒ "A_3"

")"

")" …

Readings



- For this lecture
  - Sections 6.3, 6,5
  - Chapter 13
- Coming next
  - Chapter 17