# Analysis of Algorithms
# CS 477/677

Instructor: Monica Nicolescu

Lecture 17

# 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.

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

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

# 1. The Structure of an Optimal Parenthesization

- Notation:

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

- 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 $i \leq k < j$

# 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:

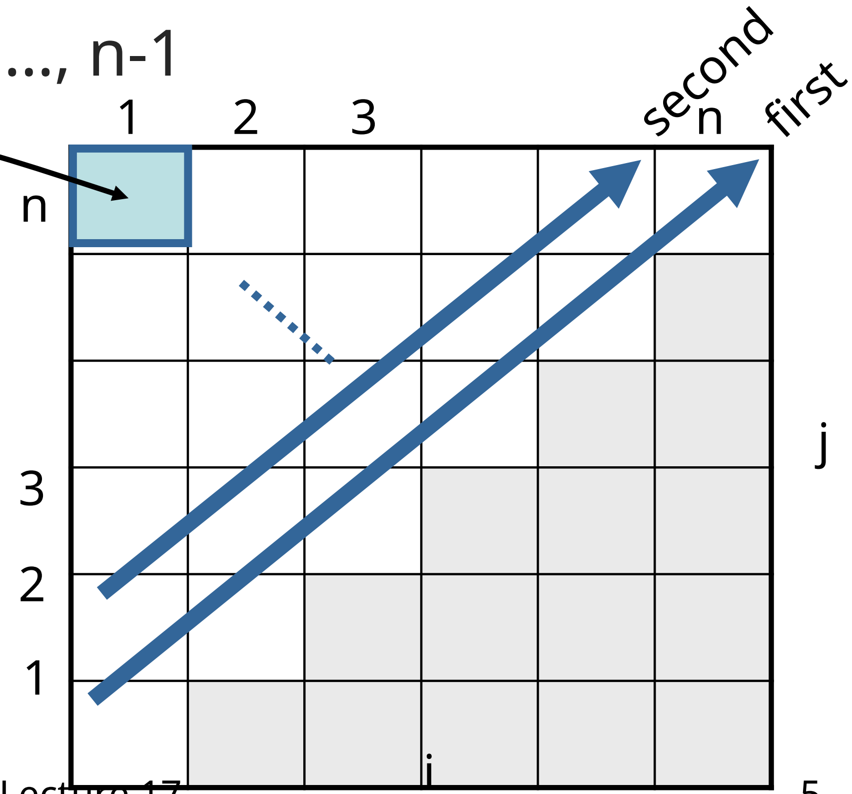
$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \leq k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\} & \text{if } i < j \end{cases}$$

# 3. Computing the Optimal Costs

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

- 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.

# Memoization

- Top-down approach with the efficiency of typical bottom-up dynamic programming approach
- Maintains an entry in a table for the solution to each subproblem
  - **memoize** the inefficient recursive top-down algorithm
- When a subproblem is first encountered its solution is computed and stored in that table
- Subsequent “calls” to the subproblem simply look up that value

# Memoized Matrix-Chain

Alg.: MEMOIZED-MATRIX-CHAIN(p)

1. n ← length[p]
2. **for** i ← 1 **to** n
3. **do for** j ← i **to** n
4. **do** m[i, j] ← ∞
5. **return** LOOKUP-CHAIN(p, 1, n)

Initialize the **m** table with large values that indicate whether the values of **m[i, j]** have been computed

Top-down approach

# Memoized Matrix-Chain

Alg.: LOOKUP-CHAIN(p, i, j)

Running time is $O(n^3)$

```
if m[i, j] < ∞
        then return m[i, j]
if i = j
   then m[i, j] ← 0
   else for k ← i to j – 1
              do q ← LOOKUP-CHAIN(p, i, k) +
                     LOOKUP-CHAIN(p, k+1, j) + p_{i-1}p_k p_j
                 if q < m[i, j]
                    then m[i, j] ← q
return m[i, j]
```

$$m[i, j] = \min_{i \le k < j} \{m[i, k] + m[k+1, j] + p_{i-1}p_k p_j\}$$

# Dynamic Progamming vs. Memoization

- Advantages of dynamic programming vs. memoized algorithms
  - No overhead for recursion
  - The regular pattern of table accesses may be used to reduce time or space requirements
- Advantages of memoized algorithms vs. dynamic programming
  - More intuitive

# Optimal Substructure - Examples

- Assembly line
  - Fastest way of going through a station j contains: the fastest way of going through station j-1 on either line
- Matrix multiplication
  - Optimal parenthesization of $A_i \bullet A_{i+1} \bullet\bullet\bullet A_j$ that splits the product between $A_k$ and $A_{k+1}$ contains:
    - an optimal solution to the problem of parenthesizing $A_{i..k}$
    - an optimal solution to the problem of parenthesizing $A_{k+1..j}$

# Parameters of Optimal Substructure

- Intuitively, the running time of a dynamic programming algorithm depends on two factors:
  - Number of subproblems overall
  - How many choices we examine for each subproblem
- Assembly line
  - $\Theta(n)$ subproblems (n stations)
  - 2 choices for each subproblem

  $\Theta(n)$ overall
- Matrix multiplication:
  - $\Theta(n^2)$ subproblems ($1 \le i \le j \le n$)
  - At most n-1 choices

  $\Theta(n^3)$ overall

# Longest Common Subsequence

- Given two sequences

  $X = \langle x_1, x_2, \ldots, x_m \rangle$

  $Y = \langle y_1, y_2, \ldots, y_n \rangle$

  find a maximum length common subsequence (LCS) of X and Y
- E.g.:

  $X = \langle A, B, C, B, D, A, B \rangle$
- Subsequence of X:
  - A subset of elements in the sequence taken in order (but not necessarily consecutive)

  $\langle A, B, D \rangle$, $\langle B, C, D, B \rangle$, etc.

# Example

X = ⟨A, B, C, B, D, A, B⟩ X = ⟨A, B, C, B, D, A, B⟩

Y = ⟨B, D, C, A, B, A⟩ Y = ⟨B, D, C, A, B, A⟩

- ⟨B, C, B, A⟩ and ⟨B, D, A, B⟩ are longest common subsequences of X and Y (length = 4)
- ⟨B, C, A⟩ , however is not a LCS of X and Y

# Brute-Force Solution

- For every subsequence of X, check whether it's a subsequence of Y
- There are $2^m$ subsequences of X to check
- Each subsequence takes $\Theta(n)$ time to check
  - scan Y for first letter, from there scan for second, and so on
- Running time: $\Theta(n2^m)$

# 1. Making the choice

$X = \langle A, B, D, E \rangle$

$Y = \langle Z, B, E \rangle$

- Choice: include one element into the common sequence (E) and solve the resulting subproblem

$X = \langle A, B, D, G \rangle$ $X = \langle A, B, D, G \rangle$

$Y = \langle Z, B, D \rangle$ $Y = \langle Z, B, D \rangle$

- Choice: exclude an element from a string and solve the resulting subproblem

# Notations

- Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$ we define the i-th prefix of X, for i = 0, 1, 2, …, m

$$X_i = \langle x_1, x_2, \ldots, x_i \rangle$$

- c[i, j] = the length of a LCS of the sequences $X_i = \langle x_1, x_2, \ldots, x_i \rangle$ and $Y_j = \langle y_1, y_2, \ldots, y_j \rangle$

# 2. A Recursive Solution

Case 1: $x_i = y_j$

e.g.: $X_i = \langle A, B, D, E \rangle$

$Y_j = \langle Z, B, E \rangle$

$$c[i, j] = c[i - 1, j - 1] + 1$$

- Append $x_i = y_j$ to the LCS of $X_{i-1}$ and $Y_{j-1}$
- Must find a LCS of $X_{i-1}$ and $Y_{j-1}$ ⇒ optimal solution to a problem includes optimal solutions to subproblems

# 2. A Recursive Solution

Case 2: $x_i \neq y_j$

e.g.: $X_i = \langle A, B, D, G \rangle$

$Y_j = \langle Z, B, D \rangle$

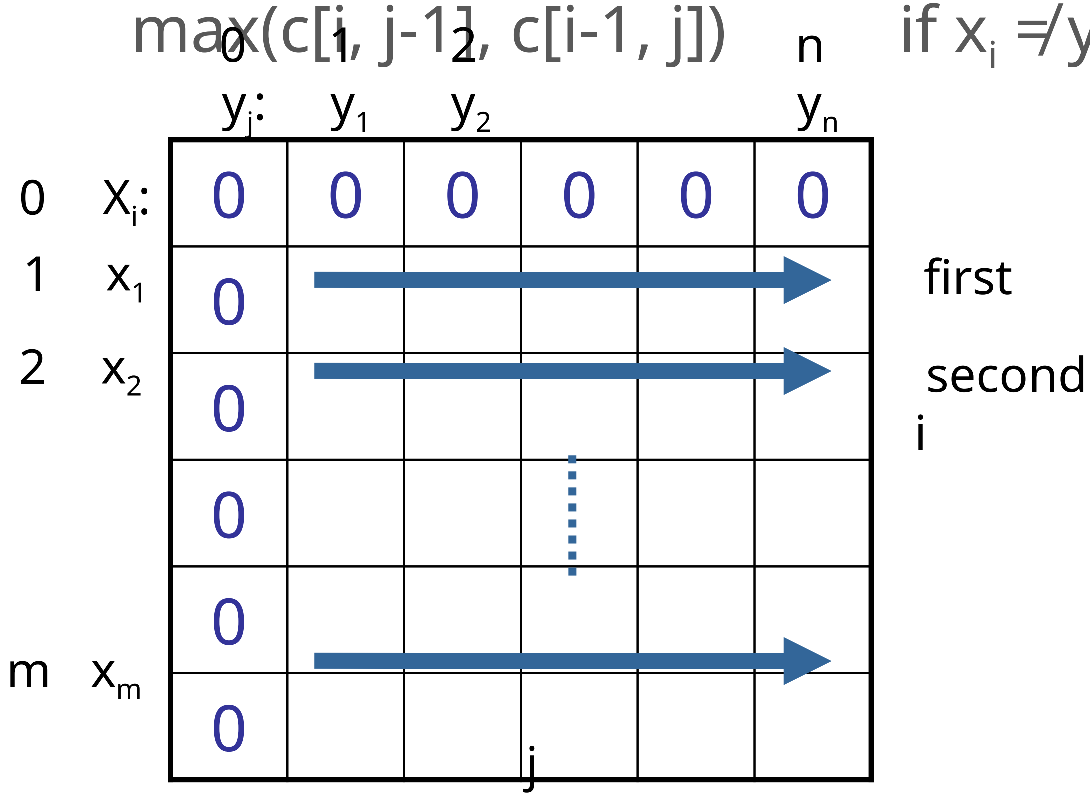
$$c[i, j] = \max \{ c[i - 1, j], c[i, j-1] \}$$

– Must solve two problems
  - find a LCS of $X_{i-1}$ and $Y_j$: $X_{i-1} = \langle A, B, D \rangle$ and $Y_j = \langle Z, B, D \rangle$
  - find a LCS of $X_i$ and $Y_{j-1}$: $X_i = \langle A, B, D, G \rangle$ and $Y_j = \langle Z, B \rangle$

- Optimal solution to a problem includes optimal solutions to subproblems

# 3. Computing the Length of the LCS

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } x_i \neq y_j \end{cases}$$



# 4. Additional Information

$$c[i, j] = \begin{cases} 0 & \text{if } i, j = 0 \\ c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } x_i \neq y_j \end{cases}$$

b & c:

| | | 0 | 1 | 2 | 3 | | n |
|---|---|---|---|---|---|---|---|
| | | $y_j$ | A | C | D | | F |
| 0 | $x_i$ | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | | | | | |
| 2 | B | 0 | | | c[i-1,j] | | |
| 3 | C | 0 | | c[i,j-1] | | | |
| | | 0 | | | | | |
| m | D | 0 | | | | | |

i

j

A matrix b[i, j]:

- For a subproblem [i, j] it tells us what choice was made to obtain the optimal value
- If $x_i = y_j$

  b[i, j] = "↖"
- Else, if

  c[i - 1, j] ≥ c[i, j-1]

  b[i, j] = " ↑ "

  else

  b[i, j] = " ←"

# LCS-LENGTH(X, Y, m, n)

```
for i ← 1 to m
    do c[i, 0] ← 0
for j ← 0 to n
    do c[0, j] ← 0
for i ← 1 to m
    do for j ← 1 to n
        do if x_i = y_j
            then c[i, j] ← c[i - 1, j - 1] + 1
                 b[i, j ] ← "↖"
            else if c[i - 1, j] ≥ c[i, j - 1]
                then c[i, j] ← c[i - 1, j]
                     b[i, j] ← "↑"
                else c[i, j] ← c[i, j - 1]
                     b[i, j] ← "←"
return c and b
```

The length of the LCS is zero if one of the sequences is empty

Case 1: $x_i = y_j$

Case 2: $x_i \neq y_j$

Running time: $\Theta(mn)$

# Example

$X = \langle A, B, C, B, D, A, B \rangle$
$Y = \langle B, D, C, A, B, A \rangle$

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } x_i \neq y_j \end{cases}$$

If $x_i = y_j$

$b[i, j] = $ " ↖ "

Else if

$c[i - 1, j] \geq c[i, j-1]$

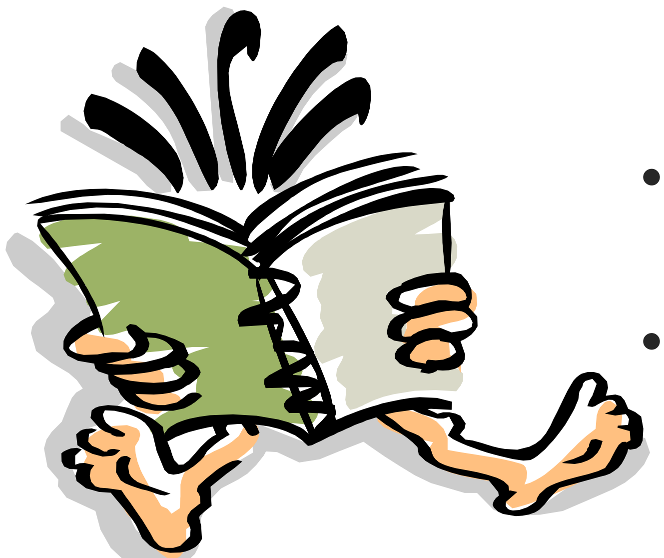
$b[i, j] = $ " ↑ "

else

$b[i, j] = $ " ← "

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| | | $y_j$ | B | D | C | A | B | A |
| 0 | $x_i$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↖ 1 | ← 1 | ↖ 1 |
| 2 | B | 0 | ↖ 1 | ← 1 | ← 1 | ↑ 1 | ↖ 2 | ← 2 |
| 3 | C | 0 | ↑ 1 | ↑ 1 | ↖ 2 | ← 2 | ↑ 2 | ↑ 2 |
| 4 | B | 0 | ↖ 1 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ 3 | ← 3 |
| 5 | D | 0 | ↑ 1 | ↖ 2 | ↑ 2 | ↑ 2 | ↑ 3 | ↑ 3 |
| 6 | A | 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↖ 3 | ↑ 3 | ↖ 4 |
| 7 | B | 0 | ↖ 1 | ↑ 2 | ↑ 2 | ↑ 3 | ↖ 4 | ↑ 4 |

# Readings



- For this lecture
  - Chapter 14
- Coming next
  - Chapter 14