DESIGN AND ANALYSIS OF ALGORITHMS

CS 4120/5120

DP – MATRIX CHAIN MULTIPLICATION

AGENDA

- Matrix chain multiplication
  - The problem
  - Apply dynamic programming to solve the problem

ELEMENTS OF DP BRIEF REVIEW

- The four elements of dynamic programming
  - Two key ingredients
    - Optimal substructure
    - Overlapping subproblems
  - Reconstructing a solution
  - Memoization

MATRIX MULTIPLICATION ALGORITHM

- The pseudocode calculates the dot-product of two ***compatible*** matrices A and B.
    - When we say A and B are ***compatible***, we are referring the dimensions of A and B satisfying the following condition.
        - A is a $p \times q$ matrix and B is a $q \times r$ matrix.

```
MATRIX-MULTIPLY $(A, B)$
1 **if** $A.columns \neq B.rows$
2         **error** “incompatible dimensions”
3 **else** let $C$ be a new $A.rows \times B.columns$ matrix
4         **for** $i = 1$ **to** $A.rows$
5                 **for** $j = 1$ **to** $B.columns$
6                         $c_{ij} = 0$
7                         **for** $k = 1$ **to** $A.columns$
8                                 $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
9         **return** $C$
```

- Particularly, line 8 of the algorithm is called a ***scalar multiplication***.

MATRIX MULTIPLICATION ALGORITHM

- The running time of the algorithm is dominated by the number of **scalar multiplications**, which is determined by
  - ______________,
  - ______________, and
  - ______________
- Suppose that A is a $p \times q$ matrix and B is a $q \times r$ matrix, the number of scalar multiplications can be calculated as ____________________.

```
MATRIX-MULTIPLY $(A, B)$
if $A.columns \neq B.rows$
    error "incompatible dimensions"
else let $C$ be a new $A.rows \times B.columns$ matrix
    for $i = 1$ to $A.rows$
        for $j = 1$ to $B.columns$
            $c_{ij} = 0$
            for $k = 1$ to $A.columns$
                $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
    return $C$
```

SCALAR MULTIPLICATION
PRACTICE

- Execute the algorithm on matrices A and B as shown below.

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \text{ and } B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix}$$

  - Matrix A is _____ × _____, and B is _____ × _____.
  - The resulting matrix C is _____ × _____.
  - The number of ***scalar multiplications*** of this particular instance is ______________________.

```
MATRIX-MULTIPLY $(A, B)$
if $A.columns \neq B.rows$
    error "incompatible dimensions"
else let $C$ be a new $A.rows \times B.columns$ matrix
    for $i = 1$ to $A.rows$
        for $j = 1$ to $B.columns$
            $c_{ij} = 0$
            for $k = 1$ to $A.columns$
                $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
    return $C$
```

MATRIX-CHAIN MULTIPLICATION

- Say we want to calculate $A_1 A_2 A_3$.
  - The dimension of A_1 is 5×5.
  - The dimension of A_2 is 5×2.
  - The dimension of A_3 is 2×3.
- Question
  - Are the three matrices $A_1, A_2, \text{and } A_3$ compatible?
  - If they are compatible, what is the dimension of the resulting matrix?
  - Are $\left((A_1 A_2) A_3\right)$ and $\left(A_1 (A_2 A_3)\right)$ equivalent to calculating $A_1 A_2 A_3$?

MATRIX-CHAIN MULTIPLICATION
CASE 1

- Say we want to calculate $A_1A_2A_3$.
    - A_1 is $5×5$, A_2 is $5×2$, A_3 is $2×3$.
- Question
    - How to use the MATRIX-MULTIPLY algorithm to compute $A_1A_2A_3$?
        - Call _____ = MATRIX-MULTIPLY (_____, _____)
            - ______ scalar multiplication, yielding a ______ matrix.
        - Call MATRIX-MULTIPLY (_____, _____)
            - _______ scalar multiplication, yielding a _____ × _____ matrix.

```
MATRIX-MULTIPLY $(A, B)$
if $A.columns \neq B.rows$
    error "incompatible dimensions"
else let $C$ be a new $A.rows \times B.columns$ matrix
    for $i = 1$ to $A.rows$
        for $j = 1$ to $B.columns$
            $c_{ij} = 0$
            for $k = 1$ to $A.columns$
                $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
    return $C$
```

MATRIX-CHAIN MULTIPLICATION
CASE 2

- Say we want to calculate $A_1A_2A_3$.
  - A_1 is 5×5, A_2 is 5×2, A_3 is 2×3.
- Question
  - How about computing $\left((A_1A_2)A_3\right)$?
    - Call _____ = MATRIX-MULTIPLY (_____, _____)
      - ______ scalar multiplication, yielding a ______ matrix.
    - Call MATRIX-MULTIPLY (_____, _____)
      - _______ scalar multiplication, yielding a _____ × _____ matrix.

MATRIX-MULTIPLY (A, B)

```
if $A.columns \neq B.rows$
    error "incompatible dimensions"
else let $C$ be a new $A.rows \times B.columns$ matrix
    for $i = 1$ to $A.rows$
        for $j = 1$ to $B.columns$
            $c_{ij} = 0$
            for $k = 1$ to $A.columns$
                $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
    return $C$
```

MATRIX-CHAIN MULTIPLICATION

CASE 3

- Say we want to calculate $A_1A_2A_3$.
  - A_1 is 5×5, A_2 is 5×2, A_3 is 2×3.
- Question
  - How about computing $\left(A_1(A_2A_3)\right)$?
    - Call _____ = MATRIX-MULTIPLY (_____, _____)
      - ______ scalar multiplication, yielding a ______ matrix.
    - Call MATRIX-MULTIPLY (_____, _____)
      - _______ scalar multiplication, yielding a _____ × _____ matrix.

```
MATRIX-MULTIPLY $(A, B)$
if $A.columns \neq B.rows$
    error "incompatible dimensions"
else let $C$ be a new $A.rows \times B.columns$ matrix
    for $i = 1$ to $A.rows$
        for $j = 1$ to $B.columns$
            $c_{ij} = 0$
            for $k = 1$ to $A.columns$
                $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
    return $C$
```

MATRIX-CHAIN MULTIPLICATION EFFICIENCY

- Previously, we determined that $A_1A_2A_3 = \left((A_1A_2)A_3\right) = \left(A_1(A_2A_3)\right)$.
  - For $A_1A_2A_3$, ___________ scalar multiplications were involved.
  - For $\left((A_1A_2)A_3\right)$, ___________ scalar multiplications were involved.
  - For $\left(A_1(A_2A_3)\right)$, ___________ scalar multiplications were involved.

- Which of the three computations is the most efficient?

MATRIX-CHAIN MULTIPLICATION

- The number of scalar multiplications varies based on *different parenthesization* of the matrices.
  - $\left((A_1A_2)A_3\right)$ vs. $\left(A_1(A_2A_3)\right)$
  - How we parenthesize a chain of matrices can have a dramatic impact on the cost of evaluating the product.

- Since matrix-multiplication is a costly process, we can **take advantage of different parenthesizations** to **minimize** the number of scalar multiplications involved.
  - Matrix multiplication is ***associative***. All parenthesizations yield the same product.

FULLY PARENTHESIZED MATRIX CHAIN

- A product of matrices is ***fully parenthesized*** if it is either a single matrix or the product of two fully parenthesized matrix products, surrounded by parentheses.
  - We can fully parenthesize the product $A_1A_2A_3A_4$ in five different ways:
    - $\left(A_1\left(A_2(A_3A_4)\right)\right)$
    - $\left(A_1\left((A_2A_3)A_4\right)\right)$
    - $\left((A_1A_2)(A_3A_4)\right)$
    - $\left(\left(A_1(A_2A_3)\right)A_4\right)$
    - $\left(\left((A_1A_2)A_3\right)A_4\right)$

THE MATRIX-CHAIN MULTIPLICATION PROBLEM

- We state the ***matrix-chain multiplication problem*** as follows
  - Given a chain $< A_1, A_2, \ldots, A_n >$ of n matrices, where for $i = 1, 2, \ldots, n$,
  - Matrix A_i has dimension $p_{i-1} \times p_i$,
  - Goal: Fully parenthesize the product $A_1 A_2 \cdots A_n$ in a way that ***minimizes*** the number of scalar multiplications.

POSSIBLE SOLUTIONS

- Brute-force
  - This is one way to go.
  - However, as the matrix chain grows longer, the number of different parenthesizations increases dramatically.
    - As a matter of fact, it grows as $\Omega(2^n)$, where n is the size of the matrix chain.
- Dynamic programming?

DYNAMIC PROGRAMMING CHECKLIST

- Here is a checklist of the qualifications of a DP problem.
    - ☐ Optimization problem
    - ☐ Two key ingredients
        - ☐ Optimal substructure
        - ☐ Overlapping subproblems

OPTIMIZATION PROBLEM

- We state the ***matrix-chain multiplication problem*** as follows
  - Given a chain $< A_1, A_2, \dots, A_n >$ of n matrices, where for $i = 1, 2, \dots, n$,
  - Matrix A_i has dimension $p_{i-1} \times p_i$,
  - Goal: Fully parenthesize the product $A_1 A_2 \cdots A_n$ in a way that ***minimizes*** the number of scalar multiplications.
- Keyword: ***minimize***

DYNAMIC PROGRAMMING CHECKLIST

- Here is a checklist of the qualifications of a DP problem.
  - ☑ Optimization problem
  - ☐ Two key ingredients
    - ☐ Optimal substructure
    - ☐ Overlapping subproblems

OPTIMAL SUBSTRUCTURE NOTATION

- The chain of matrices: $< A_1, A_2, \ldots, A_n >$ of n matrices, where for $i = 1, 2, \ldots, n$.
- Matrix A_i has dimension $p_{i-1} \times p_i$.
  - Matrix A_1 has dimension $\underline{\quad p_0 \quad}$ × $\underline{\quad p_1 \quad}$.
  - Matrix A_{10} has dimension $\underline{\quad p_9 \quad}$ × $\underline{\quad p_{10} \quad}$.
  - Matrix A_n has dimension $\underline{\quad p_{n-1} \quad}$ × $\underline{\quad p_n \quad}$.
  - The # of scalar multiplications in computing the dot-product of a $p \times q$ and a $q \times r$ matrix is $\underline{pqr}$.
  - Let us use $\boldsymbol{A_{i..j}}$ to denote the *resulting matrix* of $A_i A_{i+1} \ldots A_j$, where $i \leq j$.

OPTIMAL SUBSTRUCTURE
NOTATION (BREAKOUT CHALLENGE)

- **Challenge #1**
- Let's examine the number of scalar multiplications involved by completing the table below.
  - Matrix A_i has dimension $p_{i-1} \times p_i$.
  - The # of scalar multiplications in computing the dot-product of a $p \times q$ and a $q \times r$ matrix is pqr.
  - $\boldsymbol{A_{i..j}}$ denotes the *resulting matrix* of $A_i A_{i+1} \dots A_j$, where $i \leq j$.

Operation	**1.** $\boldsymbol{A_1 A_2}$	**2.** $\boldsymbol{A_2 A_3}$	**3.** $\boldsymbol{A_k A_{k+1}}$	**4.** $\boldsymbol{A_{i..i}}$	**5.** $\boldsymbol{A_{1..3} A_{4..7}}$	**6.** $\boldsymbol{A_{i..k} A_{k+1..j}}$
# of Scalar Multiplication						

OPTIMAL SUBSTRUCTURE
NOTATION (BREAKOUT CHALLENGE)

- **Challenge #1**
- Let’s examine the number of scalar multiplications involved by completing the table below.
  - Matrix A_i has dimension $p_{i-1} \times p_i$.
  - The # of scalar multiplications in computing the dot-product of a $p \times q$ and a $q \times r$ matrix is pqr.
  - $\boldsymbol{A_{i..j}}$ denotes the *resulting matrix* of $A_i A_{i+1} \ldots A_j$, where $i \leq j$.

Operation	**1.** $A_1 A_2$	**2.** $A_2 A_3$	**3.** $A_k A_{k+1}$	**4.** $A_{i..i}$	**5.** $A_{1..3} A_{4..7}$	**6.** $A_{i..k} A_{k+1..j}$
# of Scalar Multiplication	$p_0 p_1 p_2$	$p_1 p_2 p_3$	$p_{k-1} p_k p_{k+1}$	0	$p_0 p_3 p_7$	$p_{i-1} p_k p_j$

OPTIMAL SUBSTRUCTURE NOTATION (BREAKOUT CHALLENGE)

- **Challenge #2**
- Consider the following matrix chain $< A_1A_2A_3A_4A_5 >$ and its dimension information given in the table.

Index	0	1	2	3	4	5
p_i	4	3	2	6	4	3

- Show the mathematical expression that calculates the # of scalar multiplications of the following operation.
  - Matrix A_i has dimension $p_{i-1} \times p_i$; $\boldsymbol{A_{i..j}}$ denotes the *resulting matrix* of $A_iA_{i+1} \dots A_j$, where $i \leq j$.
  - The # of scalar multiplications in computing the dot-product of a $p \times q$ and a $q \times r$ matrix is pqr.

Operation	A_2A_3	A_3A_4	$A_{1..3}A_{4..5}$	$A_{1..4}A_5$	$A_{2..4}A_5$
# of Scalar Multiplication					

OPTIMAL SUBSTRUCTURE
NOTATION (BREAKOUT CHALLENGE)

- **Challenge #2**
- Consider the following matrix chain $< A_1A_2A_3A_4A_5 >$ and its dimension information given in the table.

Index	0	1	2	3	4	5
p_i	4	3	2	6	4	3

- Show the mathematical expression that calculates the # of scalar multiplications of the following operation.
  - Matrix A_i has dimension $p_{i-1} \times p_i$; $\boldsymbol{A_{i..j}}$ denotes the *resulting matrix* of $A_iA_{i+1} \dots A_j$, where $i \le j$.
  - The # of scalar multiplications in computing the dot-product of a $p \times q$ and a $q \times r$ matrix is pqr.

Operation	A_2A_3	A_3A_4	$A_{1..3}A_{4..5}$	$A_{1..4}A_5$	$A_{2..4}A_5$
# of Scalar Multiplication	$p_1p_2p_3 = 3 \times 2 \times 6$	$p_2p_3p_4 = 2 \times 6 \times 4$	$p_0p_3p_5 = 4 \times 6 \times 3$	$p_0p_4p_5 = 4 \times 4 \times 3$	$p_1p_4p_5 = 3 \times 4 \times 3$

DYNAMIC PROGRAMMING CHECKLIST

- Here is a checklist of the qualifications of a DP problem.
  - ☑ Optimization problem
  - ☐ Two key ingredients
    - ☐ Optimal substructure
    - ☐ Overlapping subproblems

DISCOVERING THE OPTIMAL SUBSTRUCTURE

- **General steps**
  - **Step 1**: A solution to the problem consists of making a choice.
  - **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.
  - **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.
  - **Step 4**: Show the solutions to the subproblems used within an optimal solution to the problem must themselves be optimal by using a "cut-and-paste" technique.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 1

- **Step 1**: A solution to the problem consists of making a choice.
- In this matrix-chain multiplication problem, we do need to decide how to parenthesize the matrix chain in order to get minimum number of scalar multiplications.
  - At this point, we only consider making ONE choice.
    - We do not concern ourselves with how to make subsequent choices.
  - Once we make a decision, *subproblems arise*.

$$(A_1A_2 \cdots A_kA_{k+1})(\cdots A_n)$$

$$(A_1A_2 \cdots A_k)(A_{k+1} \cdots A_n)$$

$$(A_1A_2)(\cdots A_kA_{k+1} \cdots A_n)$$

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 2

- **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.
  - At this point, you do not concern yourself with how to determine this choice.

- Suppose that parenthesize the matrix chain at matrix A_k leads to the optimal solution.
  - In other words, suppose that we can split the matrix chain between A_k and A_{k+1}, and
  - this split will lead to an ***optimal*** **parenthesization** that costs ***minimum*** scalar multiplication**.**

$$(A_1 A_2 \cdots A_k)(A_{k+1} \cdots A_n)$$

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 3

- **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.

- Given the choice of parenthesizing at A_k, _______ subproblems arise.
  - Fully parenthesize $A_1 A_2 \cdots A_k$ to minimize the number of scalar multiplications.
  - Fully parenthesize $A_{k+1} A_{k+2} \cdots A_n$ to minimize the number of scalar multiplications.

$$(A_1 A_2 \cdot \cdot \cdot A_k)(A_{k+1} \cdots A_n)$$

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 3

- **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.

- **(Cont'd) Characterize** the subproblems.
  - One sub-chain is $A_{\mathbf{1}}A_2 \cdots A_k$ that has a fixed end A_1, the other, $A_{k+1}A_{k+2} \cdots A_{\boldsymbol{n}}$ has a fixed end A_n.
  - From the two observations, we can **see that making either end fixed will lose generality**.
  - Therefore, we are going to characterize the subproblems as fully parenthesizing matrix chain

$$< A_{\boldsymbol{i}}A_{i+1} \cdots A_kA_{k+1} \cdots A_{\boldsymbol{j}} >$$

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4

- **Step 4**: Show the solutions to the subproblem used within an optimal solution to the problem must themselves be optimal by using a "cut-and-paste" technique.

- To prove this optimal substructure, we need to define new notations in addition to p.
  - Let $m[i,j]$ be **the *minimum* number** of scalar multiplications needed to compute the matrix $A_{i..j}$.
    - This definition of $m[i,j]$ indicate that $m[i,j]$ itself is THE ***OPTIMAL*** VALUE.
    - For the full problem, the lowest-cost way to compute $A_{1..n}$ would thus $\underline{\ m[1,n]\ }$.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4

- **Step 4 (Cont'd)**: The proof of the optimality of the solution to the subproblem.
  - **Step a**: Derive a **recurrence** relation.
    - Based off steps 1 ~ 3
      - Step 2: We were given the choice that leads to an ***optimal*** value, i.e., parenthesizing at A_k.
      - Step 3: We determined to characterize the subproblems as having two open ends.
    - We can derive the recurrence relation as

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

The ***minimum*** number of scalar multiplications needed to compute the matrix $A_{i..\mathbf{k}}$

The ***minimum*** number of scalar multiplications needed to compute the matrix $A_{k+1..j}$

The number of scalar multiplications needed to compute $A_{i..\mathbf{k}} \cdot A_{k+1..j}$

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4

- **Step 4 (Cont'd)**: The proof of the optimality of the solution to the subproblem.
  - **Step b**: Use a "*cut-and-paste*" technique to derive contradiction.
    - The recurrence is: $m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_k p_j$
    - The goal is to prove that $\underline{\ m[i,k]\ \ \ \ \ }$ is the ***minimum*** number of scalar multiplication needed to compute $A_{i..k}$, and $\underline{\ m[k+1,j]\ }$ $A_{k+1..j}$.
    - **Assume** $m^*[i,k]$ is the ***minimum*** number of scalar multiplication needed to compute $\underline{\ \ \ A_{i..k}\ \ \ \ \ }$, and **assume** $m^*[k+1,j]$ is the ***minimum*** number of scalar multiplication needed to compute $\underline{\ \ A_{k+1..j}\ \ \ \ }$.
    - Obviously, we have the following relations.
      - $m^*[i,k]$ $\underline{\ \ \ \ <\ \ \ \ \ }$ ($<$ or $>$) $m[i,k]$, and
      - $m^*[k+1,j]$ $\underline{\ \ \ \ <\ \ \ \ \ }$ ($<$ or $>$) $m[k+1,j]$.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4

- **Step 4 (Cont'd):** The proof of the optimality of the solution to the subproblem.
  - **Step b (Cont'd):** Use a "*cut-and-paste*" technique to derive contradiction.
    - The recurrence is: $m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
    - We can construct a **new** ***minimum*** number of scalar multiplication by cutting $\underline{\;m[i,k]\;}$ and $\underline{m[k+1,j]}$ out of the recurrence and pasting $\underline{\;m^*[i,k]\;}$ and $\underline{m^*[k+1,j]}$ to the recurrence.
    - The new recurrence, denoted by $m^*[i,j]$ is computed as $\underline{m^*[i,j] = \ m^*[i,k] + m^*[k+1,j] + p_{i-1}p_kp_j}$.
    - The relation is $m^*[i,j]$ $\underline{\quad < \quad}$ ($<$ or $>$) $m[i,j]$, which **contradicts** the definition of $m[i,j]$.
    - Therefore, $m[i,k]$ is the ***minimum*** number of scalar multiplication needed to compute $A_{i..k}$, and $m[k+1,j]$ $A_{k+1..j}$.

DISCOVERING THE OPTIMAL SUBSTRUCTURE, DONE

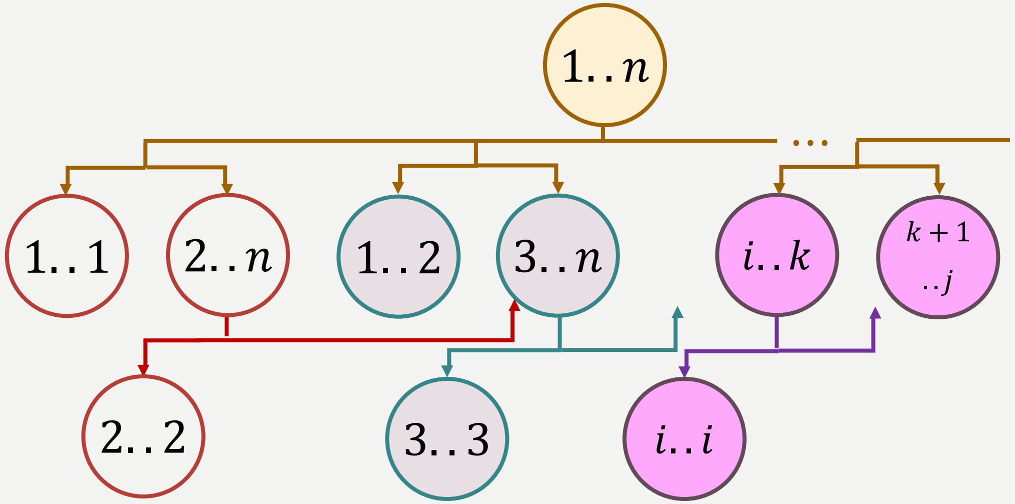
- The ***optimal substructure*** of the matrix-chain multiplication problem is as follows.
  - Suppose that to ***optimal***ly parenthesize $A_iA_{i+1} \cdots A_j$, we **split the product between A_k and A_{k+1}**.
  - Then **the way we parenthesize the "prefix" subchain $A_iA_{i+1} \cdots A_k$** within this optimal parenthesization of $A_iA_{i+1} \cdots A_j$ must be an ***optimal*** parenthesization of $A_iA_{i+1} \cdots A_k$.
  - The same observation holds for **how we parenthesize the subchain $A_{k+1}A_{k+2} \cdots A_j$** in the optimal parenthesization of $A_iA_{i+1} \cdots A_j$: it must be an ***optimal*** parenthesization of $A_{k+1}A_{k+2} \cdots A_j$.

DYNAMIC PROGRAMMING CHECKLIST

- Here is a checklist of the qualifications of a DP problem.
  - ☑ Optimization problem
  - ☐ Two key ingredients
    - ☑ Optimal substructure
    - ☐ Overlapping subproblems

DISCOVER OVERLAPPING SUBPROBLEMS

- Using subproblem graph
  - A vertex $i..j$ represents the subproblem of parenthesizing matrix chain $\mathrm{A}_i A_{i+1} \cdots A_j$.
  - A direct edge from vertex $i..j$ to $s..t$ represents determining an optimal solution for subproblem $i..j$ involves directly considering an optimal solution for subproblem $s..t$.



DYNAMIC PROGRAMMING CHECKLIST

- Here is a checklist of the qualifications of a DP problem.
    - ☑ Optimization problem
    - ☑ Two key ingredients
        - ☑ Optimal substructure
        - ☑ Overlapping subproblems

- Now we have examined the problem and know that there is a dynamic programming solution to it, we can continue with the steps to develop a dynamic programming solution.

APPLYING DP
STEP 1

- **Step 1**: Characterize the structure of an optimal solution
  - Discover the ***optimal substructure*** of the problem.

- We will continue to use the notations we defined when discovering the optimal substructure.
  - A chain $< A_1, A_2, \ldots, A_n >$ of n matrices, where for $i = 1, 2, \ldots, n.$, matrix A_i has dimension $p_{i-1} \times p_i$.
  - $A_{i..j}$ is the matrix that results from evaluating the product $A_i A_{i+1} \cdots A_j$.
  - We shall use the number of scalar multiplication to define the ***cost*** of the matrix-chain multiplication.

APPLYING DP
STEP 1 (CONT'D)

- **Step 1**: Characterize the structure of an optimal solution
    - Discover the ***optimal substructure*** of the problem.

- The ***optimal substructure*** has been defined in its discovery.
    - Suppose that to ***optimal***ly parenthesize $A_i A_{i+1} \cdots A_j$, we **split the product between $\boldsymbol{A_k}$ and $\boldsymbol{A_{k+1}}$**. Then **the way we parenthesize the "prefix" subchain $\boldsymbol{A_i A_{i+1} \cdots A_k}$** within this optimal parenthesization of $A_i A_{i+1} \cdots A_j$ must be an ***optimal*** parenthesization of $A_i A_{i+1} \cdots A_k$. The same observation holds for **how we parenthesize the subchain $\boldsymbol{A_{k+1} A_{k+2} \cdots A_j}$** in the optimal parenthesization of $A_i A_{i+1} \cdots A_j$: it must be an ***optimal*** parenthesization of $A_{k+1} A_{k+2} \cdots A_j$.

APPLYING DP
STEP 2

- **Step 2**: Recursively define the ***value*** of an optimization.
  - Take advantage of the optimal substructure to recursively compute the optimal ***value***.

- When discovering the ***optimal substructure***, we have derived a recurrence relation as follows.

$$m[i,j] = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ + \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ + \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$$

, where $m[i,j]$ is defined to be **the *minimum* number** of scalar multiplications needed to compute the matrix $A_{i..j}$.

APPLYING DP
STEP 2 (CONT'D)

- **Step 2**: Recursively define the ***value*** of an optimization.
  - Take advantage of the optimal substructure to recursively compute the optimal ***value***.

- To include the consideration of bottom-out case.

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

APPLYING DP

STEP 3 (RECURSIVE)

- **Step 3**: Compute the ***value*** of an optimal solution.
  - Design an algorithm to compute the value.
- Input
  - p is an array of dimensions.
  - i and j are the indexes of the two matrices on the two ends

```
RECURSIVE-MATRIX-CHAIN $(p, i, j)$
if $i == j$
    return $0$
$m[i, j] = \infty$
for $k = i$ to $j - 1$
    $q =$ RECURSIVE-MATRIX-CHAIN $(p, i, k) +$
            RECURSIVE-MATRIX-CHAIN $(p, k + 1, j) +$
            $p_{i-1} p_k p_j$
    if $q < m[i, j]$
        $m[i, j] = q$
return $m[i, j]$
```

APPLYING DP
STEP 3 (RECURSIVE)

- **Step 3**: Compute the ***value*** of an optimal solution.
    - Design an algorithm to compute the value.
- **Bottoms-out** case
- **Line 4 ~ 7** iteratively parenthesize $A_iA_{i+1}\cdots A_j$, then recurse to solve the subproblems for $\boldsymbol{A_iA_{i+1}\cdots A_k}$ and $\boldsymbol{A_{k+1}A_{i+2}\cdots A_j}$.

```
RECURSIVE-MATRIX-CHAIN $(p, i, j)$
1 if $i == j$
2     return $0$
3 $m[i, j] = \infty$
4 for $k = i$ to $j - 1$
5     $q =$ RECURSIVE-MATRIX-CHAIN $(p, i, k)$ +
          RECURSIVE-MATRIX-CHAIN $(p, k + 1, j)$ +
          $p_{i-1}p_kp_j$
6     if $q < m[i, j]$
7         $m[i, j] = q$
8 return $m[i, j]$
```

APPLYING DP
STEP 3 (RECURSIVE, CONT'D)

- **Step 3**: Compute the ***value*** of an optimal solution.
- The algorithm computes the ***optimal*** cost in a **top-down** strategy.
- Drawbacks
  - There is **no memoization**.
  - The running time of the algorithm is $T(n) = \boldsymbol{\Omega(2^n)}$.
    - See textbook page 385 – 386.

```
RECURSIVE-MATRIX-CHAIN $(p, i, j)$
if $i == j$
    return $0$
$m[i, j] = \infty$
for $k = i$ to $j - 1$
    $q =$ RECURSIVE-MATRIX-CHAIN $(p, i, k)$ +
        RECURSIVE-MATRIX-CHAIN $(p, k + 1, j)$ +
        $p_{i-1} p_k p_j$
    if $q < m[i, j]$
        $m[i, j] = q$
return $m[i, j]$
```

APPLYING DP
STEP 3 (MEMOIZED TOP-DOWN)

- **Step 3**: Compute the ***value*** of an optimal solution.
- Improved **top-down** method with **memoziation**
  - **Line 1** computes the length of the matrix chain
  - **Memo** is created by line 2.
  - **Initialization** done by line 3
  - **Problem solved** by line 6.
- Running time $T(n) = \boldsymbol{\theta}(\boldsymbol{n^2}) + \boldsymbol{f}(\boldsymbol{n})$
  - $\boldsymbol{f}(\boldsymbol{n})$ is the running time of the LOOKUP-CHAIN procedure.

```
MEMOIZED-MATRIX-CHAIN ($p$)
1 $n = p.length - 1$
2 let $m[1..n, 1..n]$ be a new table
3 for $i = 1$ to $n$
4     for $j = i$ to $n$
5         $m[i, j] = \infty$
6 return LOOKUP-CHAIN ($m, p, 1, n$)
```

APPLYING DP STEP 3 (LOOKUP-CHAIN VS RECURSIVE)

- **Step 3**: Compute the ***value*** of an optimal solution.

```
LOOKUP-CHAIN $(\boldsymbol{m}, \boldsymbol{p}, \boldsymbol{i}, \boldsymbol{j})$
if $m[i,j] < \infty$
    return $m[i,j]$
if $i == j$
    $m[i,j] = 0$
else for $k = i$ to $j - 1$
    $q =$ LOOKUP-CHAIN $(m, p, i, k)$ +
            LOOKUP-CHAIN $(m, p, k + 1, j)$ +
            $p_{i-1} p_k p_j$
    if $q < m[i,j]$
        $m[i,j] = q$
return $m[i,j]$
```

```
RECURSIVE-MATRIX-CHAIN $(p, i, j)$
if $i == j$
    return $0$
$m[i,j] = \infty$
for $k = i$ to $j - 1$
    $q =$ RECURSIVE-MATRIX-CHAIN $(p, i, k)$ +
            RECURSIVE-MATRIX-CHAIN $(p, k + 1, j)$ +
            $p_{i-1} p_k p_j$
    if $q < m[i,j]$
        $m[i,j] = q$
return $m[i,j]$
```

APPLYING DP STEP 3
LOOKUP-CHAIN RUNNING TIME #1

- **Step 3**: Compute the ***value*** of an optimal solution.

$\Theta(n^2)$

- Running time $\boldsymbol{f(n)} = O(n^3)$
  - Analyzed by two different types of calls made.
  - Type #**1**: Calls in which $m[i,j] = \infty$
    - There are ______ calls of this type.
      - There are ______ entries in the table of $m[i,j]$.
    - When $i \neq j$, each call of this type makes asymptotically _______ recursive calls.
    - In total, type #1 complexity is bounded by ______.

```
LOOKUP-CHAIN ($m, p, i, j$)
if $m[i,j] < \infty$
    return $m[i,j]$
if $i == j$
    $m[i,j] = 0$
else for $k = i$ to $j - 1$
    $q =$ LOOKUP-CHAIN ($m, p, i, k$) +
            LOOKUP-CHAIN ($m, p, k+1, j$) +
            $p_{i-1}p_kp_j$
    if $q < m[i,j]$
        $m[i,j] = q$
return $m[i,j]$
```

APPLYING DP STEP 3
LOOKUP-CHAIN RUNNING TIME #2

- **Step 3**: Compute the ***value*** of an optimal solution.

$\Theta(n^2)$

- Running time $\boldsymbol{f(n)} = O(n^3)$
  - Analyzed by two different types of calls made.
  - Type #**2**: Calls in which $m[i, j] < \infty$
    - Line _____ through _____ gets executed.
    - Each call takes ______ time.
    - All these calls were made as recursive calls by the calls of the type #1.

```
LOOKUP-CHAIN $(\boldsymbol{m}, \boldsymbol{p}, \boldsymbol{i}, \boldsymbol{j})$
1 if $m[i, j] < \infty$
2         return $m[i, j]$
3 if $i == j$
4         $m[i, j] = 0$
5 else for $k = i$ to $j - 1$
6         $q =$ LOOKUP-CHAIN $(m, p, i, k)$ +
                  LOOKUP-CHAIN $(m, p, k + 1, j)$ +
                  $p_{i-1}p_k p_j$
7         if $q < m[i, j]$
8                 $m[i, j] = q$
9 return $m[i, j]$
```

APPLYING DP STEP 3
LOOKUP-CHAIN RUNNING TIME

- **Step 3**: Compute the ***value*** of an optimal solution.

$\Theta(n^2)$

- Running time $\boldsymbol{f(n)} = O(n^3)$
  - Analyzed by two different types of calls made.
  - Type #**1** takes ______________ time.
  - Each of type #**2** takes ______________ time.

```
LOOKUP-CHAIN $(m, p, i, j)$
if $m[i, j] < \infty$
    return $m[i, j]$
if $i == j$
    $m[i, j] = 0$
else for $k = i$ to $j - 1$
    $q =$ LOOKUP-CHAIN $(m, p, i, k) +$
            LOOKUP-CHAIN $(m, p, k + 1, j) +$
            $p_{i-1} p_k p_j$
    if $q < m[i, j]$
        $m[i, j] = q$
return $m[i, j]$
```

APPLYING DP STEP 3 LOOKUP-CHAIN SPACE COMPLEXITY

- **Step 3**: Compute the ***value*** of an optimal solution.
- Space complexity $\boldsymbol{S(n)}$ = __________.

```
REMEMOIZED-MATRIX-CHAIN $(p)$
$n = p.length - 1$
let $m[1..n, 1..n]$ be a new table
for $i = 1$ to $n$
    for $j = i$ to $n$
        $m[i, j] = \infty$
return LOOKUP-CHAIN $(m, p, 1, n)$
```

APPLYING DP
STEP 3 (BOTTOM-UP)

- **Step 3**: Compute the ***value*** of an optimal solution.
- Input
  - An array p representing the dimension the dimensions of the matrices in the chain.
    - Array index starts at 0.

```
MATRIX-CHAIN-ORDER $(p)$
$n = p.length - 1$
let $m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tables
for $i = 1$ to $n$
    $m[i, i] = 0$
for $l = 2$ to $n$
    for $i = 1$ to $n - l + 1$
        $j = i + l - 1$
        $m[i, j] = \infty$
        for $k = i$ to $j - 1$
            $q = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$
            if $q < m[i, j]$
                $m[i, j] = q$
                $s[i, j] = k$
return $m$ and $s$
```

APPLYING DP
STEP 3 (BOTTOM-UP)

- **Step 3**: Compute the ***value*** of an optimal solution.
- MATRIX-CHAIN-ORDER algorithm
  - **Line 1** computes the length of the matrix chain
  - **Memo** is created by line 2.
    - **Solution table** also created here.
  - **Initialization** of the diagonal entry done by line 3 and 4
  - **Problem solved** by line 5 ~ 14.

```
MATRIX-CHAIN-ORDER $(p)$
1  $n = p.length - 1$
2  let $m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tables
3  for $i = 1$ to $n$
4      $m[i, i] = 0$
5  for $l = 2$ to $n$          // $l$ is the chain length
6      for $i = 1$ to $n - l + 1$
7          $j = i + l - 1$
8          $m[i, j] = \infty$
9          for $k = i$ to $j - 1$
10             $q = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$
11             if $q < m[i, j]$
12                 $m[i, j] = q$
13                 $s[i, j] = k$
14 return $m$ and $s$
```

APPLYING DP
STEP 3 (BOTTOM-UP CLOSER LOOK)

- **Step 3**: Compute the ***value*** of an optimal solution.
- MATRIX-CHAIN-ORDER algorithm
  - **Solving the problem**
    - Starting at the shortest chain possible.
    - For each length l, solve the same-length matrix chain starting at $A_1, A_3, \ldots, A_{n-1}$.
    - $l = 2$

$$\left(A_1 \quad A_2\right) A_3 \cdots\cdots A_k A_{k+1} \cdots A_{n-1} A_n$$

```
MATRIX-CHAIN-ORDER $(p)$
$n = p.length - 1$
let $m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tables
for $i = 1$ to $n$
    $m[i,i] = 0$
for $l = 2$ to $n$            // $l$ is the chain length
    for $i = 1$ to $n - l + 1$
        $j = i + l - 1$
        $m[i,j] = \infty$
        for $k = i$ to $j - 1$
            $q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j$
            if $q < m[i,j]$
                $m[i,j] = q$
                $s[i,j] = k$
return $m$ and $s$
```

APPLYING DP

STEP 3 (BOTTOM-UP RUNNING TIME)

- **Step 3**: Compute the ***value*** of an optimal solution.
- The running time of the algorithm is easily analyzed as the code is structured as triply-nested **for**-loops

```
for $l = 2$ to $n$
    for $i = 1$ to $n - l + 1$
        for $k = i$ to $j - 1$
```

- The running time $T(n) = O\ (n^3)$

```
MATRIX-CHAIN-ORDER $(p)$
$n = p.length - 1$
let $m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tables
for $i = 1$ to $n$
    $m[i,i] = 0$
for $l = 2$ to $n$          // $l$ is the chain length
    for $i = 1$ to $n - l + 1$
        $j = i + l - 1$
        $m[i,j] = \infty$
        for $k = i$ to $j - 1$
            $q = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
            if $q < m[i,j]$
                $m[i,j] = q$
                $s[i,j] = k$
return $m$ and $s$
```

APPLYING DP STEP 3 SUMMARY

- **Top-down** memoized algorithm
    - MEMOIZED-MATRIX-CHAIN (p)
    - LOOKUP-CHAIN(m, p, i, j)
    - Time complexity $T(n) = O(n^3)$
    - Space complexity $S(n) = \Theta(n^2)$
    - Neither algorithm generates a solution
    - They compute the ***optimal*** value ONLY.

- **Bottom-up** algorithm with memoization
    - MATRIX-CHAIN-ORDER (p)
    - Time complexity $T(n) = O(n^3)$
    - Space complexity $S(n) = \Theta(n^2)$
    - The algorithm above computes the the ***optimal*** value while saving the solution in an s table.

APPLYING DP

STEP 4 (PRINT-OPTIMAL-PARENS)

- **Step 4**: Construct the optimal solution from the computed information.
  - At step 3, the bottom-up algorithm MATRIX-CHAIN-ORDER (p) computes the ***optimal*** value while saving the solution in an s table.
    - We can also doctor the top-down MEMOIZED-MATRIX-CHAIN (p) and LOOKUP-CHAIN(m, p, i, j) such that they also save the solution in an s table.

```
MATRIX-CHAIN-ORDER $(p)$
$n = p.length - 1$
let $m[1..n, 1..n]$ and $\mathbf{s[1..n-1, 2..n]}$ be new tables
for $i = 1$ to $n$
    $m[i,i] = 0$
for $l = 2$ to $n$
    for $i = 1$ to $n - l + 1$
        $j = i + l - 1$
        $m[i,j] = \infty$
        for $k = i$ to $j - 1$
            $q = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
            if $q < m[i,j]$
                $m[i,j] = q$
                $\mathbf{s[i,j] = k}$
return $m$ and $s$
```

APPLYING DP
STEP 4 (PRINT-OPTIMAL-PARENS)

- **Step 4**: Construct the optimal solution from the computed information.

- Once the solution table s is computed, call PRINT-OPTIMAL-PARENS ($\underline{\boldsymbol{s}}$, $\underline{\boldsymbol{1}}$, $\underline{\boldsymbol{n}}$) to print the solution.

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

BREAKOUT PRACTICE (15 MINUTES)

matrix	A_1	A_2	A_3	A_4
dimension	3×5	5×2	2×4	4×6

- Consider a matrix chain $< A_1, A_2, A_3, A_4 >$.
  The dimension of the matrices are given in the table.
- Derive the array p that can be used as input to MATRIX-CHAIN-ORDER and PRINT-OPTIMAL-PARENS algorithms. Then run the two algorithms to complete the m and s tables.
  - Pay attention to the indexes of the two tables.

$\boldsymbol{m}$			$\boldsymbol{i}$		
		1	2	3	4
	4				
	3				
$\boldsymbol{j}$	2				
	1				

$\boldsymbol{s}$			$\boldsymbol{i}$	
		1	2	3
	4			
$\boldsymbol{j}$	3			
	2			

matrix	A_1	A_2	A_3	A_4
dimension	3×5	5×2	2×4	4×6

index i	0	1	2	3	4
p_i	3	5	2	4	6

```
MATRIX-CHAIN-ORDER ($p$)
$n = p.length - 1$
let $m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tables
for $i = 1$ to $n$
    $m[i,i] = 0$
for $l = 2$ to $n$
    for $i = 1$ to $n - l + 1$
        $j = i + l - 1$
        $m[i,j] = \infty$
        for $k = i$ to $j - 1$
            $q = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
            if $q < m[i,j]$
                $m[i,j] = q$
                $s[i,j] = k$
return $m$ and $s$
```

m			i		
		1	2	3	4
	4				
j	3				
	2				
	1				

s			i	
		1	2	3
	4			
j	3			
	2			

$l = 2 \le 4$

$i \le 4 - l + 1$	j	$k \le j - 1$	q	$< m[i,j]$?
$1 \le 3$	2	$1 \le 1$		
$2 \le 3$	3	$2 \le 2$		
$3 \le 3$	4	$3 \le 3$		

matrix	A_1	A_2	A_3	A_4
dimension	3×5	5×2	2×4	4×6

index i	0	1	2	3	4
p_i	3	5	2	4	6

MATRIX-CHAIN-ORDER (p)

```
$n = p.length - 1$
let $m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tables
for $i = 1$ to $n$
        $m[i,i] = 0$
for $l = 2$ to $n$
        for $i = 1$ to $n - l + 1$
                $j = i + l - 1$
                $m[i,j] = \infty$
                for $k = i$ to $j - 1$
                        $q = m[i,k] + m[k+1,j] + p_{i-1}p_k p_j$
                        if $q < m[i,j]$
                                $m[i,j] = q$
                                $s[i,j] = k$
return $m$ and $s$
```

m	i 1	2	3	4
j 4			48	0
3		40	0	
2	30	0		
1	0			

s	i 1	2	3
j 4			3
3		2	
2	1		

$l = 3 \le 4$

$i \le$ $4 - l + 1$	j	$k \le$ $j - 1$	q	$<$ $m[i,j]$?
$1 \le 2$	3	$1 \le 2$		
		$2 \le 2$		
$2 \le 2$	4	$2 \le 3$		
		$3 \le 3$		

matrix	A_1	A_2	A_3	A_4
dimension	3×5	5×2	2×4	4×6

index i	0	1	2	3	4
p_i	3	5	2	4	6

```
MATRIX-CHAIN-ORDER $(p)$
$n = p.length - 1$
let $m[1..n, 1..n]$ and $s[1..n-1, 2..n]$ be new tables
for $i = 1$ to $n$
        $m[i,i] = 0$
for $l = 2$ to $n$
        for $i = 1$ to $n - l + 1$
                $j = i + l - 1$
                $m[i,j] = \infty$
                for $k = i$ to $j - 1$
                        $q = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
                        if $q < m[i,j]$
                                $m[i,j] = q$
                                $s[i,j] = k$
return $m$ and $s$
```

$l = 4 \leq 4$

$i \leq$ $4 - l + 1$	j	$k \leq$ $j - 1$	q	$<$ $m[i,j]$?
$1 \leq 1$	4	$1 \leq 3$		
		$2 \leq 3$		
		$3 \leq 3$		

m i

j	1	2	3	4
4		108	48	0
3	54 ~~100~~	40	0	
2	30	0		
1	0			

s i

j	1	2	3
4		2	3
3	2 ~~1~~	2	
2	1		

matrix	A_1	A_2	A_3	A_4
dimension	3×5	5×2	2×4	4×6

→

index i	0	1	2	3	4
p_i	3	5	2	4	6

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

m	*i*			
j	1	2	3	4
4	114 ~~198~~	108	48	0
3	54 ~~100~~	40	0	
2	30	0		
1	0			

s	*i*		
j	1	2	3
4	2 ~~1~~	2	3
3	2 ~~1~~	2	
2	1		

Recursions	Print
PRINT-OPTIMAL-PARENS $(s, 1, 4)$	(
PRINT-OPTIMAL-PARENS $(s, 1, s[1, 4] = 2)$	(
PRINT-OPTIMAL-PARENS $(s, 1, s[1, 2] = 1)$	A1
PRINT-OPTIMAL-PARENS $(s, s[1,2] + 1 = 2, 2)$	A2
	)
PRINT-OPTIMAL-PARENS $(s, s[1, 4] + 1 = 3, 4)$	(
PRINT-OPTIMAL-PARENS $(s, 3, s[3,4] = 3)$	A3
PRINT-OPTIMAL-PARENS $(s, s[3,4] + 1, 4)$	A4
	)
	)

The output: ((A1A2)(A3A4))

NEXT UP
LONGEST-COMMON-SUBSEQUENCE

REFERENCE

- Screenshots are taken from the textbook.