Dynamic Programming - Matrix Chain Parenthesization

COMS10017 - (Object-Oriented Programming and) Algorithms

Dr Christian Konrad

Matrix Multiplication

Problem: MATRIX-MULTIPLICATION

1. **Input:** Matrices A, B with $A.columns = B.rows$
2. **Output:** Matrix product $A \times B$

Matrix Multiplication

Problem: MATRIX-MULTIPLICATION

1. **Input:** Matrices A, B with $A.columns = B.rows$
2. **Output:** Matrix product $A \times B$

Example:

Matrix Multiplication

Problem: MATRIX-MULTIPLICATION

1. **Input:** Matrices A, B with $A.columns = B.rows$
2. **Output:** Matrix product $A \times B$

Example:

$$p\overset{q}{\begin{pmatrix} 2 & 3 \\ 1 & 0 \\ 2 & 6 \\ 0 & 9 \end{pmatrix}} \times {}^{q}\overset{r}{\begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 0 \end{pmatrix}} = \overset{r}{\begin{pmatrix} 6 & 2 & 4 \\ 0 & 1 & 2 \\ 12 & 2 & 4 \\ 18 & 0 & 0 \end{pmatrix}} p$$

Matrix Multiplication

Problem: MATRIX-MULTIPLICATION

1. **Input:** Matrices A, B with $A.columns = B.rows$
2. **Output:** Matrix product $A \times B$

Example:

$$p\overset{q}{\begin{pmatrix} 2 & 3 \\ 1 & 0 \\ 2 & 6 \\ 0 & 9 \end{pmatrix}} \times {}^{q}\overset{r}{\begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 0 \end{pmatrix}} = \overset{r}{\begin{pmatrix} 6 & 2 & 4 \\ 0 & 1 & 2 \\ 12 & 2 & 4 \\ 18 & 0 & 0 \end{pmatrix}}p$$

Notation: $p \times q$ matrix: p rows and q columns

Matrix Multiplication

Problem: MATRIX-MULTIPLICATION

1. **Input:** Matrices A, B with $A.columns = B.rows$
2. **Output:** Matrix product $A \times B$

Example:

$$p \overset{q}{\begin{pmatrix} 2 & 3 \\ 1 & 0 \\ 2 & 6 \\ 0 & 9 \end{pmatrix}} \times q \overset{r}{\begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 0 \end{pmatrix}} = \overset{r}{\begin{pmatrix} 6 & 2 & 4 \\ 0 & 1 & 2 \\ 12 & 2 & 4 \\ 18 & 0 & 0 \end{pmatrix}} p$$

Notation: $p \times q$ matrix: p rows and q columns

- $p \times q$ matrix times $q \times r$ matrix gives a $p \times r$ matrix

Matrix Multiplication

Problem: MATRIX-MULTIPLICATION

1. **Input:** Matrices A, B with $A.columns = B.rows$
2. **Output:** Matrix product $A \times B$

Example:

$$p\overset{q}{\begin{pmatrix} 2 & 3 \\ 1 & 0 \\ 2 & 6 \\ 0 & 9 \end{pmatrix}} \times q\overset{r}{\begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 0 \end{pmatrix}} = \overset{r}{\begin{pmatrix} 6 & 2 & 4 \\ 0 & 1 & 2 \\ 12 & 2 & 4 \\ 18 & 0 & 0 \end{pmatrix}}p$$

Notation: $p \times q$ matrix: p rows and q columns

- $p \times q$ matrix times $q \times r$ matrix gives a $p \times r$ matrix
- $(A \times B)_{i,j}$ = row i of A times column j of B

Algorithm for Matrix-Multiplication

Algorithm: $(A \times B)_{i,j} =$ row i of A times column j of B

Require: Matrices A, B with $A.columns = B.rows$
  Let C be a new $A.rows \times B.columns$ matrix
  for $i \leftarrow 1 \ldots A.rows$ **do**
    for $j \leftarrow 1 \ldots B.columns$ **do**
      $C_{ij} \leftarrow 0$
      for $k \leftarrow 1 \ldots A.columns$ **do**
        $C_{ij} \leftarrow C_{ij} + A_{ik} \cdot B_{kj}$
  return C

Algorithm $\textsc{Matrix-Multiply}(A, B)$

Algorithm for Matrix-Multiplication

Algorithm: $(A \times B)_{i,j}$ = row i of A times column j of B

```
Require: Matrices A, B with A.columns = B.rows
  Let C be a new A.rows × B.columns matrix
  for i ← 1 ... A.rows do
    for j ← 1 ... B.columns do
      C_ij ← 0
      for k ← 1 ... A.columns do
        C_ij ← C_ij + A_ik · B_kj
  return C
```

Algorithm MATRIX-MULTIPLY(A, B)

Runtime:

Algorithm for Matrix-Multiplication

Algorithm: $(A \times B)_{i,j} =$ row i of A times column j of B

```
Require: Matrices A, B with A.columns = B.rows
  Let C be a new A.rows × B.columns matrix
  for i ← 1 ... A.rows do
    for j ← 1 ... B.columns do
      C_ij ← 0
      for k ← 1 ... A.columns do
        C_ij ← C_ij + A_ik · B_kj
  return C
```

Algorithm $\textsc{Matrix-Multiply}(A, B)$

Runtime:

- Three nested loops: $O(A.rows \cdot B.columns \cdot A.columns)$

Algorithm for Matrix-Multiplication

Algorithm: $(A \times B)_{i,j}$ = row i of A times column j of B

```
Require: Matrices A, B with A.columns = B.rows
  Let C be a new A.rows × B.columns matrix
  for i ← 1 . . . A.rows do
    for j ← 1 . . . B.columns do
      C_ij ← 0
      for k ← 1 . . . A.columns do
        C_ij ← C_ij + A_ik · B_kj
  return C
```

Algorithm $\textsc{Matrix-Multiply}(A, B)$

Runtime:

- Three nested loops: $O(A.rows \cdot B.columns \cdot A.columns)$
- Number of Multiplications: $A.rows \cdot B.columns \cdot A.columns$

Algorithm: $(A \times B)_{i,j} =$ row i of A times column j of B

Require: Matrices A, B with $A.columns = B.rows$
  Let C be a new $A.rows \times B.columns$ matrix
  for $i \leftarrow 1 \ldots A.rows$ **do**
    for $j \leftarrow 1 \ldots B.columns$ **do**
      $C_{ij} \leftarrow 0$
      for $k \leftarrow 1 \ldots A.columns$ **do**
        $C_{ij} \leftarrow C_{ij} + A_{ik} \cdot B_{kj}$
  return C

Algorithm $\textsc{Matrix-Multiply}(A, B)$

Runtime:

- Three nested loops: $O(A.rows \cdot B.columns \cdot A.columns)$
- Number of Multiplications: $A.rows \cdot B.columns \cdot A.columns$
- Multiplying two $n \times n$ matrices: runtime $O(n^3)$

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$
- 1969: Strassen $O(n^{2.8074})$ (divide-and-conquer)

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$
- 1969: Strassen $O(n^{2.8074})$ (divide-and-conquer)
- 1990: Coppersmith-Winograd $O(n^{2.3755})$

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$
- 1969: Strassen $O(n^{2.8074})$ (divide-and-conquer)
- 1990: Coppersmith-Winograd $O(n^{2.3755})$
- 2010: Stothers $O(n^{2.374})$

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$
- 1969: Strassen $O(n^{2.8074})$ (divide-and-conquer)
- 1990: Coppersmith-Winograd $O(n^{2.3755})$
- 2010: Stothers $O(n^{2.374})$
- 2011: Virginia Williams $O(n^{2.3728642})$

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$
- 1969: Strassen $O(n^{2.8074})$ (divide-and-conquer)
- 1990: Coppersmith-Winograd $O(n^{2.3755})$
- 2010: Stothers $O(n^{2.374})$
- 2011: Virginia Williams $O(n^{2.3728642})$
- 2014: Le Gall $O(n^{2.3728639})$

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$
- 1969: Strassen $O(n^{2.8074})$ (divide-and-conquer)
- 1990: Coppersmith-Winograd $O(n^{2.3755})$
- 2010: Stothers $O(n^{2.374})$
- 2011: Virginia Williams $O(n^{2.3728642})$
- 2014: Le Gall $O(n^{2.3728639})$
- 2020: Alman, Williams $O(n^{2.3728596})$

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$
- 1969: Strassen $O(n^{2.8074})$ (divide-and-conquer)
- 1990: Coppersmith-Winograd $O(n^{2.3755})$
- 2010: Stothers $O(n^{2.374})$
- 2011: Virginia Williams $O(n^{2.3728642})$
- 2014: Le Gall $O(n^{2.3728639})$
- 2020: Alman, Williams $O(n^{2.3728596})$
- 2022: Duan, Wu, Zhou $O(n^{2.37188})$

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$
- 1969: Strassen $O(n^{2.8074})$ (divide-and-conquer)
- 1990: Coppersmith-Winograd $O(n^{2.3755})$
- 2010: Stothers $O(n^{2.374})$
- 2011: Virginia Williams $O(n^{2.3728642})$
- 2014: Le Gall $O(n^{2.3728639})$
- 2020: Alman, Williams $O(n^{2.3728596})$
- 2022: Duan, Wu, Zhou $O(n^{2.37188})$

Important Problem:

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$
- 1969: Strassen $O(n^{2.8074})$ (divide-and-conquer)
- 1990: Coppersmith-Winograd $O(n^{2.3755})$
- 2010: Stothers $O(n^{2.374})$
- 2011: Virginia Williams $O(n^{2.3728642})$
- 2014: Le Gall $O(n^{2.3728639})$
- 2020: Alman, Williams $O(n^{2.3728596})$
- 2022: Duan, Wu, Zhou $O(n^{2.37188})$

Important Problem:

- Many algorithms rely on fast matrix multiplication

Background: Faster Matrix Multiplication

History: Multiplying two $n \times n$ matrices

- before 1969: $O(n^3)$
- 1969: Strassen $O(n^{2.8074})$ (divide-and-conquer)
- 1990: Coppersmith-Winograd $O(n^{2.3755})$
- 2010: Stothers $O(n^{2.374})$
- 2011: Virginia Williams $O(n^{2.3728642})$
- 2014: Le Gall $O(n^{2.3728639})$
- 2020: Alman, Williams $O(n^{2.3728596})$
- 2022: Duan, Wu, Zhou $O(n^{2.37188})$

Important Problem:

- Many algorithms rely on fast matrix multiplication
- Better bound for matrix multiplication improves many algorithms

The Matrix-chain Multiplication Problem

Problem: MATRIX-CHAIN-MULTIPLICATION

The Matrix-chain Multiplication Problem

Problem: MATRIX-CHAIN-MULTIPLICATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \ldots, A_n$

The Matrix-chain Multiplication Problem

Problem: MATRIX-CHAIN-MULTIPLICATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \ldots, A_n$
2. **Output:** The product $A_1 \times A_2 \times A_3 \times \cdots \times A_n$

The Matrix-chain Multiplication Problem

Problem: MATRIX-CHAIN-MULTIPLICATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \ldots, A_n$
2. **Output:** The product $A_1 \times A_2 \times A_3 \times \cdots \times A_n$

Discussion:

The Matrix-chain Multiplication Problem

Problem: MATRIX-CHAIN-MULTIPLICATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \ldots, A_n$
2. **Output:** The product $A_1 \times A_2 \times A_3 \times \cdots \times A_n$

Discussion:

- $A_i.columns = A_{i+1}.rows$ for every $1 \leq i < n$

The Matrix-chain Multiplication Problem

Problem: MATRIX-CHAIN-MULTIPLICATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \ldots, A_n$
2. **Output:** The product $A_1 \times A_2 \times A_3 \times \cdots \times A_n$

Discussion:

- $A_i.columns = A_{i+1}.rows$ for every $1 \leq i < n$
- Assume A_i has dimension $p_{i-1} \times p_i$, for vector $p[0 \ldots n]$

The Matrix-chain Multiplication Problem

Problem: MATRIX-CHAIN-MULTIPLICATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \ldots, A_n$
2. **Output:** The product $A_1 \times A_2 \times A_3 \times \cdots \times A_n$

Discussion:

- $A_i.columns = A_{i+1}.rows$ for every $1 \leq i < n$
- Assume A_i has dimension $p_{i-1} \times p_i$, for vector $p[0 \ldots n]$
- Matrix product is associative:

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

The Matrix-chain Multiplication Problem

Problem: MATRIX-CHAIN-MULTIPLICATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \ldots, A_n$
2. **Output:** The product $A_1 \times A_2 \times A_3 \times \cdots \times A_n$

Discussion:

- $A_i.columns = A_{i+1}.rows$ for every $1 \leq i < n$
- Assume A_i has dimension $p_{i-1} \times p_i$, for vector $p[0 \ldots n]$
- Matrix product is associative:

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

Exploit Associativity:

The Matrix-chain Multiplication Problem

Problem: MATRIX-CHAIN-MULTIPLICATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \dots, A_n$
2. **Output:** The product $A_1 \times A_2 \times A_3 \times \cdots \times A_n$

Discussion:

- $A_i.columns = A_{i+1}.rows$ for every $1 \leq i < n$
- Assume A_i has dimension $p_{i-1} \times p_i$, for vector $p[0 \dots n]$
- Matrix product is associative:

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

Exploit Associativity: Parenthesize $A_1 \times A_2 \times A_3 \times \dots A_n$ so as to minimize the number of scalar multiplications (and thus the runtime)

Order matters

Example:

Order matters

Example: Three matrices A_1, A_2, A_3 with dimensions

$$A_1 : 10 \times 100 \quad A_2 : 100 \times 5 \quad A_3 : 5 \times 50$$

$(p_0 = 10, p_1 = 100, p_2 = 5, p_3 = 50)$

Order matters

Example: Three matrices A_1, A_2, A_3 with dimensions

$$A_1 : 10 \times 100 \quad A_2 : 100 \times 5 \quad A_3 : 5 \times 50$$

$(p_0 = 10, p_1 = 100, p_2 = 5, p_3 = 50)$

Computation of $(A_1 \times A_2) \times A_3$:

Order matters

Example: Three matrices A_1, A_2, A_3 with dimensions

$$A_1 : 10 \times 100 \quad A_2 : 100 \times 5 \quad A_3 : 5 \times 50$$

$(p_0 = 10, p_1 = 100, p_2 = 5, p_3 = 50)$

Computation of $(A_1 \times A_2) \times A_3$:

- $A_1 \times A_2 = A_{12}$ requires $10 \cdot 100 \cdot 5 = 5000$ multiplications

Order matters

Example: Three matrices A_1, A_2, A_3 with dimensions

$$A_1 : 10 \times 100 \quad A_2 : 100 \times 5 \quad A_3 : 5 \times 50$$

$(p_0 = 10, p_1 = 100, p_2 = 5, p_3 = 50)$

Computation of $(A_1 \times A_2) \times A_3$:

- $A_1 \times A_2 = A_{12}$ requires $10 \cdot 100 \cdot 5 = 5000$ multiplications
- $A_{12} \times A_3$ requires $10 \cdot 5 \cdot 50 = 2500$ multiplications

Order matters

Example: Three matrices A_1, A_2, A_3 with dimensions

$$A_1 : 10 \times 100 \quad A_2 : 100 \times 5 \quad A_3 : 5 \times 50$$

$(p_0 = 10, p_1 = 100, p_2 = 5, p_3 = 50)$

Computation of $(A_1 \times A_2) \times A_3$:

- $A_1 \times A_2 = A_{12}$ requires $10 \cdot 100 \cdot 5 = 5000$ multiplications
- $A_{12} \times A_3$ requires $10 \cdot 5 \cdot 50 = 2500$ multiplications
- Total: 7500 multiplications

Order matters

Example: Three matrices A_1, A_2, A_3 with dimensions

$$A_1 : 10 \times 100 \quad A_2 : 100 \times 5 \quad A_3 : 5 \times 50$$

$(p_0 = 10, p_1 = 100, p_2 = 5, p_3 = 50)$

Computation of $(A_1 \times A_2) \times A_3$:

- $A_1 \times A_2 = A_{12}$ requires $10 \cdot 100 \cdot 5 = 5000$ multiplications
- $A_{12} \times A_3$ requires $10 \cdot 5 \cdot 50 = 2500$ multiplications
- Total: 7500 multiplications

Computation of $A_1 \times (A_2 \times A_3)$:

Order matters

Example: Three matrices A_1, A_2, A_3 with dimensions

$$A_1 : 10 \times 100 \quad A_2 : 100 \times 5 \quad A_3 : 5 \times 50$$

$(p_0 = 10, p_1 = 100, p_2 = 5, p_3 = 50)$

Computation of $(A_1 \times A_2) \times A_3$:

- $A_1 \times A_2 = A_{12}$ requires $10 \cdot 100 \cdot 5 = 5000$ multiplications
- $A_{12} \times A_3$ requires $10 \cdot 5 \cdot 50 = 2500$ multiplications
- Total: 7500 multiplications

Computation of $A_1 \times (A_2 \times A_3)$:

- $A_2 \times A_3 = A_{23}$ requires $100 \cdot 5 \cdot 50 = 25000$ multiplications

Order matters

Example: Three matrices A_1, A_2, A_3 with dimensions

$$A_1 : 10 \times 100 \quad A_2 : 100 \times 5 \quad A_3 : 5 \times 50$$

$(p_0 = 10, p_1 = 100, p_2 = 5, p_3 = 50)$

Computation of $(A_1 \times A_2) \times A_3$:

- $A_1 \times A_2 = A_{12}$ requires $10 \cdot 100 \cdot 5 = 5000$ multiplications
- $A_{12} \times A_3$ requires $10 \cdot 5 \cdot 50 = 2500$ multiplications
- Total: 7500 multiplications

Computation of $A_1 \times (A_2 \times A_3)$:

- $A_2 \times A_3 = A_{23}$ requires $100 \cdot 5 \cdot 50 = 25000$ multiplications
- $A_1 \times A_{23}$ requires $10 \cdot 100 \cdot 50 = 50000$ multiplications

Order matters

Example: Three matrices A_1, A_2, A_3 with dimensions

$$A_1 : 10 \times 100 \quad A_2 : 100 \times 5 \quad A_3 : 5 \times 50$$

$(p_0 = 10, p_1 = 100, p_2 = 5, p_3 = 50)$

Computation of $(A_1 \times A_2) \times A_3$:

- $A_1 \times A_2 = A_{12}$ requires $10 \cdot 100 \cdot 5 = 5000$ multiplications
- $A_{12} \times A_3$ requires $10 \cdot 5 \cdot 50 = 2500$ multiplications
- Total: 7500 multiplications

Computation of $A_1 \times (A_2 \times A_3)$:

- $A_2 \times A_3 = A_{23}$ requires $100 \cdot 5 \cdot 50 = 25000$ multiplications
- $A_1 \times A_{23}$ requires $10 \cdot 100 \cdot 50 = 50000$ multiplications
- Total: 75000 multiplications

The Matrix-Chain-Parenthesization Problem

Problem: MATRIX-CHAIN-PARENTHESIZATION

The Matrix-Chain-Parenthesization Problem

Problem: MATRIX-CHAIN-PARENTHESIZATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \ldots, A_n$

The Matrix-Chain-Parenthesization Problem

Problem: MATRIX-CHAIN-PARENTHESIZATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \dots, A_n$
2. **Output:** A parenthesization of $A_1 \times A_2 \times A_3 \times \cdots \times A_n$ that minimizes the number of scalar multiplications

The Matrix-Chain-Parenthesization Problem

Problem: MATRIX-CHAIN-PARENTHESIZATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \ldots, A_n$
2. **Output:** A parenthesization of $A_1 \times A_2 \times A_3 \times \cdots \times A_n$ that minimizes the number of scalar multiplications

How many Parenthesizations $P(n)$ are there?

The Matrix-Chain-Parenthesization Problem

Problem: MATRIX-CHAIN-PARENTHESIZATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \ldots, A_n$
2. **Output:** A parenthesization of $A_1 \times A_2 \times A_3 \times \cdots \times A_n$ that minimizes the number of scalar multiplications

How many Parenthesizations $P(n)$ are there?

- We write: A_{ij} for the product $A_i \times A_{i+1} \times \cdots \times A_j$

The Matrix-Chain-Parenthesization Problem

Problem: MATRIX-CHAIN-PARENTHESIZATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \dots, A_n$
2. **Output:** A parenthesization of $A_1 \times A_2 \times A_3 \times \cdots \times A_n$ that minimizes the number of scalar multiplications

How many Parenthesizations $P(n)$ are there?

- We write: A_{ij} for the product $A_i \times A_{i+1} \times \cdots \times A_j$
- There is a final matrix multiplication: $A_{1k} \times A_{(k+1)n}$, for some $1 \leq k \leq n-1$.

The Matrix-Chain-Parenthesization Problem

Problem: MATRIX-CHAIN-PARENTHESIZATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \dots, A_n$
2. **Output:** A parenthesization of $A_1 \times A_2 \times A_3 \times \cdots \times A_n$ that minimizes the number of scalar multiplications

How many Parenthesizations $P(n)$ are there?

- We write: A_{ij} for the product $A_i \times A_{i+1} \times \cdots \times A_j$
- There is a final matrix multiplication: $A_{1k} \times A_{(k+1)n}$, for some $1 \le k \le n-1$. Hence:

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \ , \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \ . \end{cases}$$

The Matrix-Chain-Parenthesization Problem

Problem: MATRIX-CHAIN-PARENTHESIZATION

1. **Input:** A sequence (chain) of n matrices $A_1, A_2, A_3, \dots, A_n$
2. **Output:** A parenthesization of $A_1 \times A_2 \times A_3 \times \cdots \times A_n$ that minimizes the number of scalar multiplications

How many Parenthesizations $P(n)$ are there?

- We write: A_{ij} for the product $A_i \times A_{i+1} \times \cdots \times A_j$
- There is a final matrix multiplication: $A_{1k} \times A_{(k+1)n}$, for some $1 \le k \le n-1$. Hence:

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \ , \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \ge 2 \ . \end{cases}$$

Example: Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

$P(3)$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

$$P(3) \;=\; \sum_{k=1}^{2} P(k)P(n-k)$$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

$$P(3) \;=\; \sum_{k=1}^{2} P(k)P(n-k) = P(1)P(2) + P(2)P(1)$$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

$$P(3) \;=\; \sum_{k=1}^{2} P(k)P(n-k) = P(1)P(2) + P(2)P(1) = 2$$

$P(4)$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

$$P(3) = \sum_{k=1}^{2} P(k)P(n-k) = P(1)P(2) + P(2)P(1) = 2$$

$$P(4) = \sum_{k=1}^{3} P(k)P(n-k)$$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

$$\begin{aligned} P(3) &= \sum_{k=1}^{2} P(k)P(n-k) = P(1)P(2) + P(2)P(1) = 2 \\ P(4) &= \sum_{k=1}^{3} P(k)P(n-k) = P(1)P(3) + P(2)P(2) + P(3)P(1) \end{aligned}$$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

$$\begin{aligned}
P(3) &= \sum_{k=1}^{2} P(k)P(n-k) = P(1)P(2) + P(2)P(1) = 2 \\
P(4) &= \sum_{k=1}^{3} P(k)P(n-k) = P(1)P(3) + P(2)P(2) + P(3)P(1) \\
&= P(3) + 1 + P(3)
\end{aligned}$$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

$$\begin{aligned} P(3) &= \sum_{k=1}^{2} P(k)P(n-k) = P(1)P(2) + P(2)P(1) = 2 \\ P(4) &= \sum_{k=1}^{3} P(k)P(n-k) = P(1)P(3) + P(2)P(2) + P(3)P(1) \\ &= P(3) + 1 + P(3) = 2P(3) + 1 \end{aligned}$$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

$$\begin{aligned}
P(3) &= \sum_{k=1}^{2} P(k)P(n-k) = P(1)P(2) + P(2)P(1) = 2 \\
P(4) &= \sum_{k=1}^{3} P(k)P(n-k) = P(1)P(3) + P(2)P(2) + P(3)P(1) \\
&= P(3) + 1 + P(3) = 2P(3) + 1 = 5 \ .
\end{aligned}$$

Number of Parenthesizations

Example (continued): Four matrices A_1, A_2, A_3, A_4

$$A_1 \times A_{24} \quad A_{12} \times A_{34} \quad A_{13} \times A_4$$

$$\begin{aligned} P(3) &= \sum_{k=1}^{2} P(k)P(n-k) = P(1)P(2) + P(2)P(1) = 2 \\ P(4) &= \sum_{k=1}^{3} P(k)P(n-k) = P(1)P(3) + P(2)P(2) + P(3)P(1) \\ &= P(3) + 1 + P(3) = 2P(3) + 1 = 5 \ . \end{aligned}$$

1. $A_1 \times ((A_2 \times A_3) \times A_4)$
2. $A_1 \times ((A_2 \times (A_3 \times A_4))$
3. $(A_1 \times A_2) \times (A_3 \times A_4)$
4. $((A_1 \times A_2) \times A_3) \times A_4$
5. $(A_1 \times (A_2 \times A_3)) \times A_4$

A Bound on the Number of Parenthesizations:

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \ , \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \ . \end{cases}$$

$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, \ldots$

Number of Parenthesizations (2)

A Bound on the Number of Parenthesizations:

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \ , \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \ . \end{cases}$$

$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, \ldots$

- It can be seen that there are $\Omega(2^n)$ possibilities

Number of Parenthesizations (2)

A Bound on the Number of Parenthesizations:

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \ , \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \ . \end{cases}$$

$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, \dots$

- It can be seen that there are $\Omega(2^n)$ possibilities
- An efficient algorithm thus cannot try out all possibilities

Number of Parenthesizations (2)

A Bound on the Number of Parenthesizations:

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \ , \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \ . \end{cases}$$

$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, \dots$

- It can be seen that there are $\Omega(2^n)$ possibilities
- An efficient algorithm thus cannot try out all possibilities

Dynamic Programming!

Optimal Substructure

Optimal Substructure

Optimal Substructure

We say that a problem P exhibits *optimal substructure* if:

An optimal solution to P contains within it optimal solutions to subproblems of P.

Optimal Substructure

Optimal Substructure

We say that a problem P exhibits *optimal substructure* if:

An optimal solution to P contains within it optimal solutions to subproblems of P.

Optimal Substructure in Matrix-Chain-Parenthesization

Optimal Substructure

Optimal Substructure

We say that a problem P exhibits *optimal substructure* if:

An optimal solution to P contains within it optimal solutions to subproblems of P.

Optimal Substructure in Matrix-Chain-Parenthesization

- Consider optimal solution to instance of size n

Optimal Substructure

Optimal Substructure

We say that a problem P exhibits *optimal substructure* if:

An optimal solution to P contains within it optimal solutions to subproblems of P.

Optimal Substructure in Matrix-Chain-Parenthesization

- Consider optimal solution to instance of size n
- Suppose that last product is $A_{1k} \times A_{(k+1)n}$

Optimal Substructure

Optimal Substructure

We say that a problem P exhibits *optimal substructure* if:

An optimal solution to P contains within it optimal solutions to subproblems of P.

Optimal Substructure in Matrix-Chain-Parenthesization

- Consider optimal solution to instance of size n
- Suppose that last product is $A_{1k} \times A_{(k+1)n}$
- Then the optimal solution contains optimal parenthesizations of $A_1 \times A_2 \times \cdots \times A_k$ and $A_{k+1} \times A_{k+2} \times \ldots A_n$

Optimal Substructure

Optimal Substructure

We say that a problem P exhibits *optimal substructure* if:

An optimal solution to P contains within it optimal solutions to subproblems of P.

Optimal Substructure in Matrix-Chain-Parenthesization

- Consider optimal solution to instance of size n
- Suppose that last product is $A_{1k} \times A_{(k+1)n}$
- Then the optimal solution contains optimal parenthesizations of $A_1 \times A_2 \times \cdots \times A_k$ and $A_{k+1} \times A_{k+2} \times \ldots A_n$

  Proof.

Optimal Substructure

Optimal Substructure

We say that a problem P exhibits *optimal substructure* if:

An optimal solution to P contains within it optimal solutions to subproblems of P.

Optimal Substructure in Matrix-Chain-Parenthesization

- Consider optimal solution to instance of size n
- Suppose that last product is $A_{1k} \times A_{(k+1)n}$
- Then the optimal solution contains optimal parenthesizations of $A_1 \times A_2 \times \cdots \times A_k$ and $A_{k+1} \times A_{k+2} \times \ldots A_n$

  Proof. Suppose it did not contain optimal parenthesizations of $A_1 \times A_2 \times \cdots \times A_k$ and of $A_{k+1} \times A_{k+2} \times \ldots A_n$.

Optimal Substructure

Optimal Substructure

We say that a problem P exhibits *optimal substructure* if:

An optimal solution to P contains within it optimal solutions to subproblems of P.

Optimal Substructure in Matrix-Chain-Parenthesization

- Consider optimal solution to instance of size n
- Suppose that last product is $A_{1k} \times A_{(k+1)n}$
- Then the optimal solution contains optimal parenthesizations of $A_1 \times A_2 \times \cdots \times A_k$ and $A_{k+1} \times A_{k+2} \times \ldots A_n$

  Proof. Suppose it did not contain optimal parenthesizations of $A_1 \times A_2 \times \cdots \times A_k$ and of $A_{k+1} \times A_{k+2} \times \ldots A_n$. Then picking optimal parenthesizations of the two subproblems would give better solution to initial instance. □

Recursive Solution

Optimal Solution to Subproblem:

Recursive Solution

Optimal Solution to Subproblem:

- $m[i,j]$: minimum number of scalar multiplications needed to compute $A_i \times A_{i+1} \times \cdots \times A_j = A_{ij}$

Recursive Solution

Optimal Solution to Subproblem:

- $m[i,j]$: minimum number of scalar multiplications needed to compute $A_i \times A_{i+1} \times \cdots \times A_j = A_{ij}$
- Observe that $m[i,i] = 0$ (chain consists of single matrix A_i)

Recursive Solution

Optimal Solution to Subproblem:

- $m[i,j]$: minimum number of scalar multiplications needed to compute $A_i \times A_{i+1} \times \cdots \times A_j = A_{ij}$
- Observe that $m[i,i] = 0$ (chain consists of single matrix A_i)
- Suppose $j > i$. Suppose last multiplication in optimal solution is: $A_{ik} \times A_{(k+1)j}$, for some k

Recursive Solution

Optimal Solution to Subproblem:

- $m[i,j]$: minimum number of scalar multiplications needed to compute $A_i \times A_{i+1} \times \cdots \times A_j = A_{ij}$
- Observe that $m[i,i] = 0$ (chain consists of single matrix A_i)
- Suppose $j > i$. Suppose last multiplication in optimal solution is: $A_{ik} \times A_{(k+1)j}$, for some k
- Then: cost of multiplying $A_{ik} \times A_{(k+1)j}$

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

(A_{ik}: $p_{i-1} \times p_k$ matrix, $A_{(k+1)j}$: $p_k \times p_j$ matrix)

Optimal Solution to Subproblem:

- $m[i,j]$: minimum number of scalar multiplications needed to compute $A_i \times A_{i+1} \times \cdots \times A_j = A_{ij}$
- Observe that $m[i,i] = 0$ (chain consists of single matrix A_i)
- Suppose $j > i$. Suppose last multiplication in optimal solution is: $A_{ik} \times A_{(k+1)j}$, for some k
- Then:

  cost of multiplying $A_{ik} \times A_{(k+1)j}$

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

  (A_{ik}: $p_{i-1} \times p_k$ matrix, $A_{(k+1)j}$: $p_k \times p_j$ matrix)
- Since we do not know k, we try out all possibilities and choose the best solution:

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

Computing the Optimal Costs

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

Algorithmic Considerations:

Computing the Optimal Costs

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

Algorithmic Considerations:

- As in Pole-Cutting, we could implement this recursive formula directly. → exponential runtime

Computing the Optimal Costs

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

Algorithmic Considerations:

- As in Pole-Cutting, we could implement this recursive formula directly. $\rightarrow$ exponential runtime
- Instead, we compute the table $m[i,j]$ bottom up

Computing the Optimal Costs

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

Algorithmic Considerations:

- As in Pole-Cutting, we could implement this recursive formula directly. $\rightarrow$ exponential runtime
- Instead, we compute the table $m[i,j]$ bottom up
- Observe that there are less than n^2 subproblems $m[i,j]$ (i and j take values in $\{1,\dots,n\}$)

Computing the Optimal Costs

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

Algorithmic Considerations:

- As in Pole-Cutting, we could implement this recursive formula directly. $\rightarrow$ exponential runtime
- Instead, we compute the table $m[i,j]$ bottom up
- Observe that there are less than n^2 subproblems $m[i,j]$ (i and j take values in $\{1, \dots, n\}$)
- We will see that computing one value $m[i,j]$ takes $O(n)$ time

Computing the Optimal Costs

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

Algorithmic Considerations:

- As in Pole-Cutting, we could implement this recursive formula directly. $\rightarrow$ exponential runtime
- Instead, we compute the table $m[i,j]$ bottom up
- Observe that there are less than n^2 subproblems $m[i,j]$ (i and j take values in $\{1,\dots,n\}$)
- We will see that computing one value $m[i,j]$ takes $O(n)$ time
- This yields an $O(n^3)$ time algorithm

Dynamic Programming Algorithm

Require: Integer n, vector of dimensions of matrices p so that matrix A_i has dimensions $p_{i-1} \times p_i$

Let $m[1 \ldots n, 1 \ldots n]$ be a new array
for $i \leftarrow 1 \ldots n$ **do**
  $m[i,i] \leftarrow 0$
for $\ell \leftarrow 2 \ldots n$ **do** {chain length}
  for $i \leftarrow 1 \ldots n - \ell + 1$ **do** {left position}
    $j \leftarrow i + \ell - 1$ {right position}
    $m[i,j] \leftarrow \infty$
    for $k \leftarrow i \ldots j-1$ **do**
      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\}$
return m

Algorithm Matrix-Chain-Value(n, p)

Require: Integer n, vector of dimensions of matrices p so that matrix A_i has dimensions $p_{i-1} \times p_i$

Let $m[1 \dots n, 1 \dots n]$ be a new array
for $i \leftarrow 1 \dots n$ **do**
  $m[i,i] \leftarrow 0$
for $\ell \leftarrow 2 \dots n$ **do** {chain length}
  for $i \leftarrow 1 \dots n - \ell + 1$ **do** {left position}
    $j \leftarrow i + \ell - 1$ {right position}
    $m[i,j] \leftarrow \infty$
    for $k \leftarrow i \dots j - 1$ **do**
      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\}$
return m

Algorithm MATRIX-CHAIN-VALUE(n, p)

Runtime: $O(n^3)$

Dynamic Programming Algorithm

Require: Integer n, vector of dimensions of matrices p so that matrix A_i has dimensions $p_{i-1} \times p_i$

Let $m[1 \dots n, 1 \dots n]$ be a new array

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

  $m[i,i] \leftarrow 0$

for $\ell \leftarrow 2 \dots n$ **do** {chain length}

  for $i \leftarrow 1 \dots n - \ell + 1$ **do** {left position}

    $j \leftarrow i + \ell - 1$ {right position}

    $m[i,j] \leftarrow \infty$

    for $k \leftarrow i \dots j-1$ **do**

      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\}$

return m

Algorithm MATRIX-CHAIN-VALUE(n, p)

Runtime: $O(n^3)$ (by evaluating $\sum_{\ell=2}^{n} \sum_{i=1}^{n-\ell+1} \sum_{k=1}^{i+\ell-2} O(1)$)

Runtime Evaluation

Useful Formula:

$$\sum_{i=a}^{b} 1 = b - a + 1$$

Runtime Evaluation

Useful Formula:

$$\sum_{i=a}^{b} 1 = b - a + 1$$

$$\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} O(1)$$

Runtime Evaluation

Useful Formula:

$$\sum_{i=a}^{b} 1 = b - a + 1$$

$$\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} O(1) = O(1) \cdot \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} 1$$

Runtime Evaluation

Useful Formula:

$$\sum_{i=a}^{b} 1 = b - a + 1$$

$$\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} O(1) = O(1) \cdot \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} 1$$

$$\leq \quad O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} \sum_{k=1}^{n} 1$$

Runtime Evaluation

Useful Formula:

$$\sum_{i=a}^{b} 1 = b - a + 1$$

$$\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} O(1) = O(1) \cdot \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} 1$$

$$\leq \quad O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} \sum_{k=1}^{n} 1 = O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} n$$

Runtime Evaluation

Useful Formula:

$$\sum_{i=a}^{b} 1 = b - a + 1$$

$$\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} O(1) = O(1) \cdot \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} 1$$

$$\leq \quad O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} \sum_{k=1}^{n} 1 = O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} n = O(1) \cdot n \sum_{l=1}^{n} \sum_{i=1}^{n} 1$$

Runtime Evaluation

Useful Formula:

$$\sum_{i=a}^{b} 1 = b - a + 1$$

$$\begin{aligned}
\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} O(1) &= O(1) \cdot \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} 1 \\
&\leq O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} \sum_{k=1}^{n} 1 = O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} n = O(1) \cdot n \sum_{l=1}^{n} \sum_{i=1}^{n} 1 \\
&= O(1) \cdot n \sum_{l=1}^{n} n
\end{aligned}$$

Runtime Evaluation

Useful Formula:

$$\sum_{i=a}^{b} 1 = b - a + 1$$

$$\begin{aligned}
\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} O(1) &= O(1) \cdot \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} 1 \\
&\leq O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} \sum_{k=1}^{n} 1 = O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} n = O(1) \cdot n \sum_{l=1}^{n} \sum_{i=1}^{n} 1 \\
&= O(1) \cdot n \sum_{l=1}^{n} n = O(1) \cdot n^2 \sum_{l=1}^{n} 1
\end{aligned}$$

Runtime Evaluation

Useful Formula:

$$\sum_{i=a}^{b} 1 = b - a + 1$$

$$\begin{aligned}
\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} O(1) &= O(1) \cdot \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} 1 \\
&\leq O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} \sum_{k=1}^{n} 1 = O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} n = O(1) \cdot n \sum_{l=1}^{n} \sum_{i=1}^{n} 1 \\
&= O(1) \cdot n \sum_{l=1}^{n} n = O(1) \cdot n^2 \sum_{l=1}^{n} 1 = O(1) \cdot n^2 \cdot n = O(1) n^3
\end{aligned}$$

Runtime Evaluation

Useful Formula:

$$\sum_{i=a}^{b} 1 = b - a + 1$$

$$\begin{aligned}
\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} O(1) &= O(1) \cdot \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=1}^{i+l-2} 1 \\
&\leq O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} \sum_{k=1}^{n} 1 = O(1) \cdot \sum_{l=1}^{n} \sum_{i=1}^{n} n = O(1) \cdot n \sum_{l=1}^{n} \sum_{i=1}^{n} 1 \\
&= O(1) \cdot n \sum_{l=1}^{n} n = O(1) \cdot n^2 \sum_{l=1}^{n} 1 = O(1) \cdot n^2 \cdot n = O(1) n^3 \\
&= O(n^3) \ .
\end{aligned}$$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1				
2				
3				
4				

for $i \leftarrow 1 \ldots n$ **do**
  $m[i, i] \leftarrow 0$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2		0		
3			0	
4				0

for $i \leftarrow 1 \ldots n$ **do**
  $m[i, i] \leftarrow 0$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2		0		
3			0	
4				0

for $l \leftarrow 2 \dots n$ **do**
  for $i \leftarrow 1 \dots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i,j] \leftarrow \infty$
    for $k \leftarrow i \dots j - 1$ **do**
      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\}$

$l = 2, i = 1, j = 2$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3			0	
4				0

for $l \leftarrow 2 \dots n$ **do**
  for $i \leftarrow 1 \dots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i,j] \leftarrow \infty$
    for $k \leftarrow i \dots j - 1$ **do**
      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\}$

$l = 2, i = 1, j = 2$

$$m[1,2] = m[1,1] + m[2,2] + p_0 p_1 p_2 = 0 + 0 + 3 \cdot 7 \cdot 6 = 126$$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3			0	
4				0

for $l \leftarrow 2 \ldots n$ **do**
  for $i \leftarrow 1 \ldots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i,j] \leftarrow \infty$
    for $k \leftarrow i \ldots j - 1$ **do**
      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\}$

$l = 2, i = 2, j = 3$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3		84	0	
4				0

for $l \leftarrow 2 \ldots n$ **do**
  for $i \leftarrow 1 \ldots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i, j] \leftarrow \infty$
    for $k \leftarrow i \ldots j - 1$ **do**
      $m[i, j] \leftarrow \min\{m[i, j], m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j\}$

$l = 2, i = 2, j = 3$

$$m[2, 3] = m[2, 2] + m[3, 3] + p_1 p_2 p_3 = 0 + 0 + 7 \cdot 6 \cdot 2 = 84$$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3		84	0	
4				0

for $l \leftarrow 2 \dots n$ **do**
  for $i \leftarrow 1 \dots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i, j] \leftarrow \infty$
    for $k \leftarrow i \dots j - 1$ **do**
      $m[i, j] \leftarrow \min\{m[i, j], m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j\}$

$l = 2, i = 3, j = 4$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3		84	0	
4			108	0

for $l \leftarrow 2 \ldots n$ **do**
  for $i \leftarrow 1 \ldots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i,j] \leftarrow \infty$
    for $k \leftarrow i \ldots j - 1$ **do**
      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\}$

$l = 2, i = 3, j = 4$

$$m[3,4] = m[3,3] + m[4,4] + p_2 p_3 p_4 = 0 + 0 + 6 \cdot 2 \cdot 9 = 108$$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3		84	0	
4			108	0

for $l \leftarrow 2 \dots n$ **do**
  for $i \leftarrow 1 \dots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i, j] \leftarrow \infty$
    for $k \leftarrow i \dots j - 1$ **do**
      $m[i, j] \leftarrow \min\{m[i, j], m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j\}$

$l = 3, i = 1, j = 3$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3	106	84	0	
4			108	0

for $l \leftarrow 2 \dots n$ **do**
  for $i \leftarrow 1 \dots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i, j] \leftarrow \infty$
    for $k \leftarrow i \dots j - 1$ **do**
      $m[i, j] \leftarrow \min\{m[i, j], m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j\}$

$l = 3, i = 1, j = 3$

$$m[1,1] + m[2,3] + p_0 p_1 p_3 = 0 + 84 + 3 \cdot 7 \cdot 2 = 84 + 42 = 106$$
$$m[1,2] + m[3,3] + p_0 p_2 p_3 = 126 + 0 + 3 \cdot 6 \cdot 2 = 126 + 36 = 162$$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3	106	84	0	
4			108	0

for $l \leftarrow 2 \ldots n$ **do**
  for $i \leftarrow 1 \ldots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i,j] \leftarrow \infty$
    for $k \leftarrow i \ldots j - 1$ **do**
      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\}$

$l = 3, i = 2, j = 4$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3	106	84	0	
4		210	108	0

for $l \leftarrow 2 \dots n$ **do**
  for $i \leftarrow 1 \dots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i,j] \leftarrow \infty$
    for $k \leftarrow i \dots j - 1$ **do**
      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1}p_kp_j\}$

$l = 3, i = 2, j = 4$

$$m[2,2] + m[3,4] + p_1p_2p_4 = 0 + 108 + 7 \cdot 6 \cdot 9 = 108 + 378 = 486$$
$$m[2,3] + m[4,4] + p_1p_3p_4 = 84 + 0 + 7 \cdot 2 \cdot 9 = 84 + 36 = 210$$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3	106	84	0	
4		210	108	0

for $l \leftarrow 2 \dots n$ **do**
  for $i \leftarrow 1 \dots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i,j] \leftarrow \infty$
    for $k \leftarrow i \dots j - 1$ **do**
      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\}$

$l = 4, i = 1, j = 4$

Example $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

	1	2	3	4
1	0			
2	126	0		
3	106	84	0	
4	160	210	108	0

for $l \leftarrow 2 \ldots n$ **do**
  for $i \leftarrow 1 \ldots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i,j] \leftarrow \infty$
    for $k \leftarrow i \ldots j - 1$ **do**
      $m[i,j] \leftarrow \min\{m[i,j], m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\}$

$$m[1,1] + m[2,4] + p_0 p_1 p_4 = 0 + 210 + 3 \cdot 7 \cdot 9 = 399$$
$$m[1,2] + m[3,4] + p_0 p_2 p_4 = 126 + 108 + 3 \cdot 6 \cdot 9 = 396$$
$$m[1,3] + m[4,4] + p_0 p_3 p_4 = 106 + 0 + 3 \cdot 2 \cdot 9 = 160$$

Optimal Solution of Example

Example: $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

Optimal Solution of Example

Example: $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

- Algorithm outputs value of optimal solution: $m[1, 4] = 160$

Optimal Solution of Example

Example: $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

- Algorithm outputs value of optimal solution: $m[1, 4] = 160$
- We would like to know the optimal parenthesization as well

$$((A_1 \times A_2) \times A_3) \times A_4$$

Optimal Solution of Example

Example: $n = 4$ and $p = 3 \quad 7 \quad 6 \quad 2 \quad 9$

- Algorithm outputs value of optimal solution: $m[1,4] = 160$
- We would like to know the optimal parenthesization as well

$$((A_1 \times A_2) \times A_3) \times A_4$$

$\rightarrow$ Modify algorithm to keep track of parameters that give minimum in array s

Keep Track of Optimal Choices

Require: Integer n, vector of dimensions of matrices p so that matrix A_i has dimensions $p_{i-1} \times p_i$

```
Let m[1...n, 1...n] be a new array
for i ← 1...n do
    m[i, i] ← 0
for l ← 2...n do {chain length}
    for i ← 1...n − l + 1 do {left position}
        j ← i + l − 1 {right position}
        m[i, j] ← ∞
        for k ← i...j − 1 do
            m[i, j] ← min{m[i, j], m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j}
return m, s
```

Algorithm MATRIX-CHAIN-VALUE(n, p)

Keep Track of Optimal Choices

Require: Integer n, vector of dimensions of matrices p so that matrix A_i has dimensions $p_{i-1} \times p_i$

Let $m[1 \dots n, 1 \dots n]$ and $s[1 \dots n, 2 \dots n]$ be new arrays

for $i \leftarrow 1 \dots n$ **do**
  $m[i, i] \leftarrow 0$
for $l \leftarrow 2 \dots n$ **do** {chain length}
  for $i \leftarrow 1 \dots n - l + 1$ **do** {left position}
    $j \leftarrow i + l - 1$ {right position}
    $m[i, j] \leftarrow \infty$
    for $k \leftarrow i \dots j - 1$ **do**
      $q \leftarrow m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$
      if $q < m[i, j]$ **then**
        $m[i, j] \leftarrow q$
        $s[i, j] \leftarrow k$
return m

Algorithm MATRIX-CHAIN-ORDER(A, B)

Print Optimal Parenthesization

Using s to find Optimal Parenthesization

```
Require: Array s, positions i, j
  if i = j then
    print "A_i"
  else
    print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
    print ")"
```

Algorithm PRINT-OPTIMAL-PARENS(s, i, j)

Call PRINT-OPTIMAL-PARENS$(s, 1, n)$ to obtain parenthesization