Kourosh Davoudi
kourosh@ontariotechu.ca

Lecture 5: Dynamic Programming I

CSCI 3070U: Design and Analysis of Algorithms

Learning Outcomes

- Dynamic Programming (DP):
  - What is DP?
  - Why do we need it?
- Case Studies:
  - Fibonacci Series revisit !
  - Matrix Chain Multiplication

Motivation: Fibonacci Series

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ \ldots\ .$$

```
FIB(n)
if n == 0 or n == 1
        return 1
else
        return FIB(n − 1) + FIB(n − 2)
```

$$T(0) = c_1$$
$$T(1) = c_2$$
$$T(n) = T(n-1) + T(n-2) + c_3$$

Motivation: Fibonacci Series

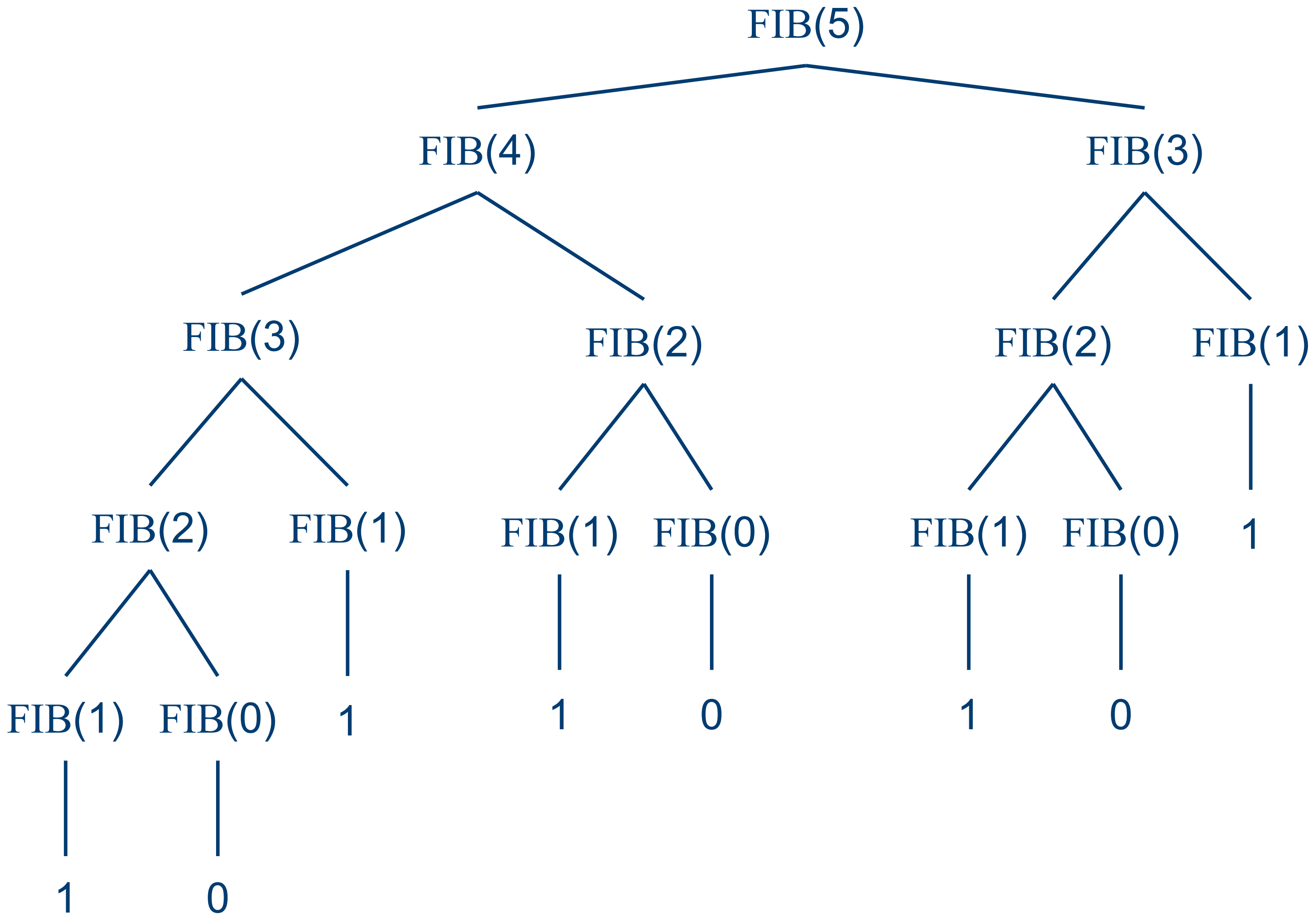
$$T(0) = T(1) = c$$
$$T(n) = T(n-1) + T(n-2) + c$$

$$T(n-2) \leq T(n-1)$$

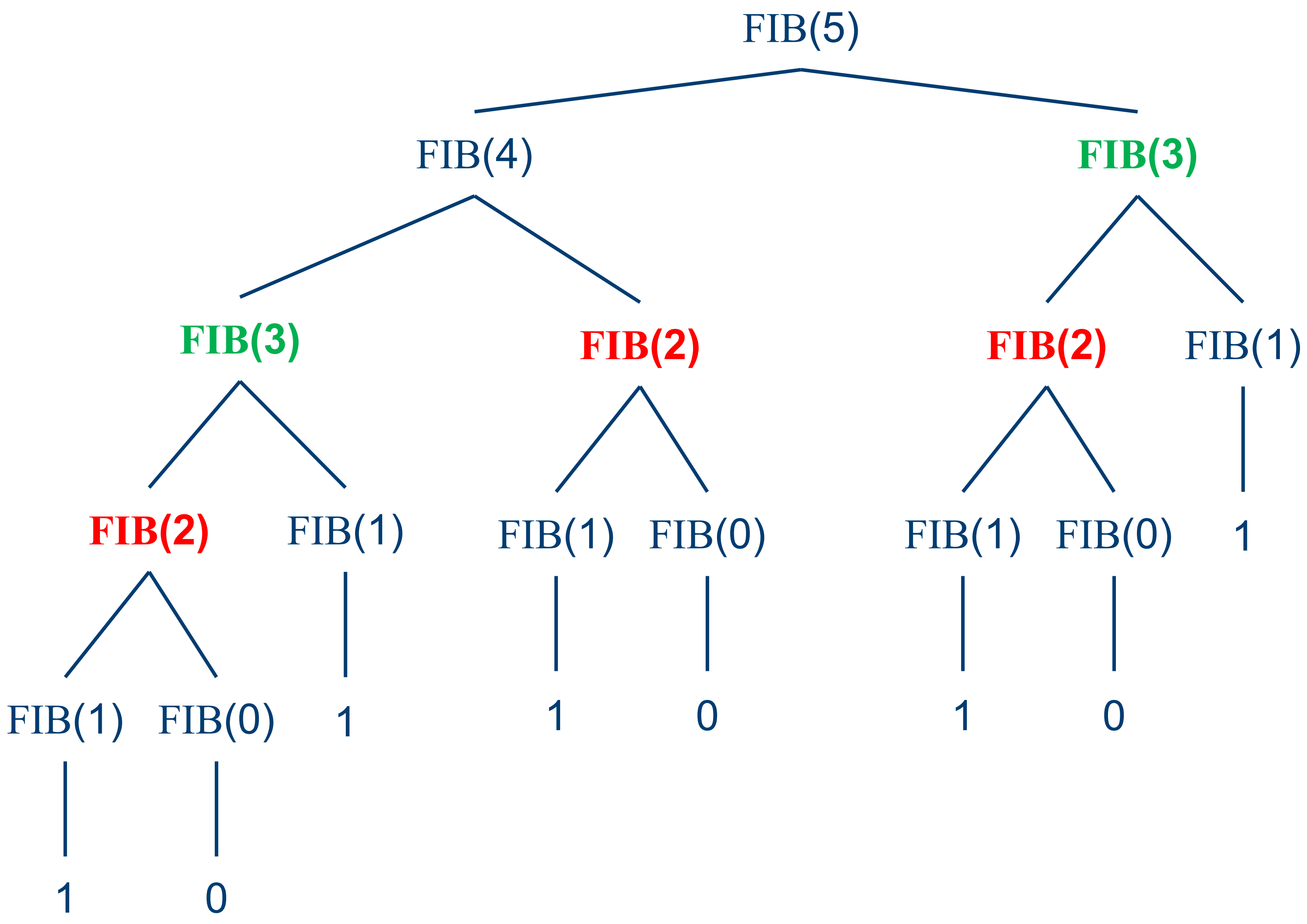
$$\begin{aligned}
T(n) &\leq 2\ T(n-1) + c \\
&\leq 2(2T(n-2) + c) + c = 2^2T(n-2) + 2c + c \\
&\leq 2^3T(n-3) + 2^2c + 2c + c \\
&\dots \\
&\leq 2^kT(n-k) + 2^{k-1}c + \cdots + 2^2c + 2c + c \\
&\leq 2^nc + (2^n - 1)c
\end{aligned}$$

$$T(n) = O(2^n)$$

Motivation: Fibonacci Series



Motivation: Fibonacci Series



Case Study: Fibonacci Series

```
FIBONACCI(n)
  let fib[0 . . n] be a new array
  fib[0] = fib[1] = 1
  for i = 2 to n
      fib[i] = fib[i − 1] + fib[i − 2]
  return fib[n]
```

$$T(n) = \Theta(n)$$

Dynamic Programming Foundation

- Idea: Previously computed subproblem solutions are stored
  - Dynamic programming often involves a space-time trade-off
- Storing computed solutions is called memoization
- Each time a computation needs to occur, check
  - if it exists in our table, Yes: Use it
  - No: Compute it, and store it in the table

When DP?

- A problem can be broken down into overlapping subproblems
  - For example FIB(3) and FIB(4) both need FIB(2)
- The solution to a subproblem does not change
- Dynamic programming often involves a space-time trade-off

Case Study: Matrix Chain Multiplication

- Recall from Linear Algebra that any sequence of matrices $M_1, M_2, \ldots M_n$ can be multiplied by grouping any two adjacent matrices
  - Matrix multiplication is associative

$$(M_1 M_2)\, M_3 = M_1\, (M_2 M_3)$$

- Does it make a difference which one we choose, when computing the result?

$$(M_1 M_2)\, M_3 \;\;?\;\; M_1\, (M_2 M_3)$$

Case Study: Matrix Chain Multiplication

- Background:
  - How many multiplications are needed for the following matrix multiplication?

$$\begin{bmatrix} \\ p \times q \\ \\ \end{bmatrix} \times \begin{bmatrix} q \times r \\ \\ \end{bmatrix}$$

Case Study: Matrix Chain Multiplication

- Background:

```
MATRIX-MULTIPLY(A, B)
if A.columns ≠ B.rows
    error "incompatible dimensions"
else let C be a new A.rows × 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 · b_kj
    return C
```

Assume:

A: p $\times$ q matrix

B: q $\times$ r matrix

$p = A.rows$

$q = A.columns = B.rows$

$r = B.column$

(A B) multiplication complexity is

$$\Theta(pqr)$$

Case Study: Matrix Chain Multiplication

- **Example:** Consider multiplying the following:

  - M_1: 10x100
  - M_2: 100x5
  - M_3: 5x50

$$(M_1M_2)\ M_3 \quad ? \quad M_1\ (M_2M_3)$$

Case Study: Matrix Chain Multiplication

M_1: 10x100

M_2: 100x5

M_3: 5x50

- Now, consider the following parenthesizations:

  - $(M_1M_2)\ M_3$:
    - $M_1\ M_2$: 10x100x5 = 5000 multiplications
    - $(M_1\ M_2)\ M_3$: 10x5x50 = **2500**

7500

  - $M_1\ (M_2M_3)$:
    - $M_2\ M_3$: 100x5x50 = 25000 multiplications
    - $M_1(M_2M_3)$: 10x100x50 = **50000**

75000

Problem Formulation

- We pick as our subproblems the problems of determining the minimum cost of parenthesizing

$$A_i A_{i+1} .. A_j \qquad 1 \le i \le j \le n$$

  - $A_i : p_{i\text{-}1} \times p_i$
- Let $m[i, j]$ be the minimum number of scalar multiplications needed to compute the matrix $A_{i..j}$
- Our goal is to find

$$m[1,n]$$

Problem Formulation

- We pick as our subproblems the problems of determining the minimum cost of parenthesizing

$$A_i A_{i+1} .. A_j \qquad 1 \leq i \leq j \leq n$$

  - $A_i : p_{i\text{-}1} \times p_i$

- Recursive Formulation:

$$(A_i A_k)\,(A_{k+1} .. A_j)$$

$$p_{i\text{-}1} \times p_k \qquad p_k \times p_j$$

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

Problem Formulation

- We pick as our subproblems the problems of determining the minimum cost of parenthesizing

$$A_i\, A_{i+1} \,..\, A_j \qquad 1 \le i \le j \le n$$

  - $A_i : p_{i\text{-}1} \times p_i$

- Recursive Formulation:

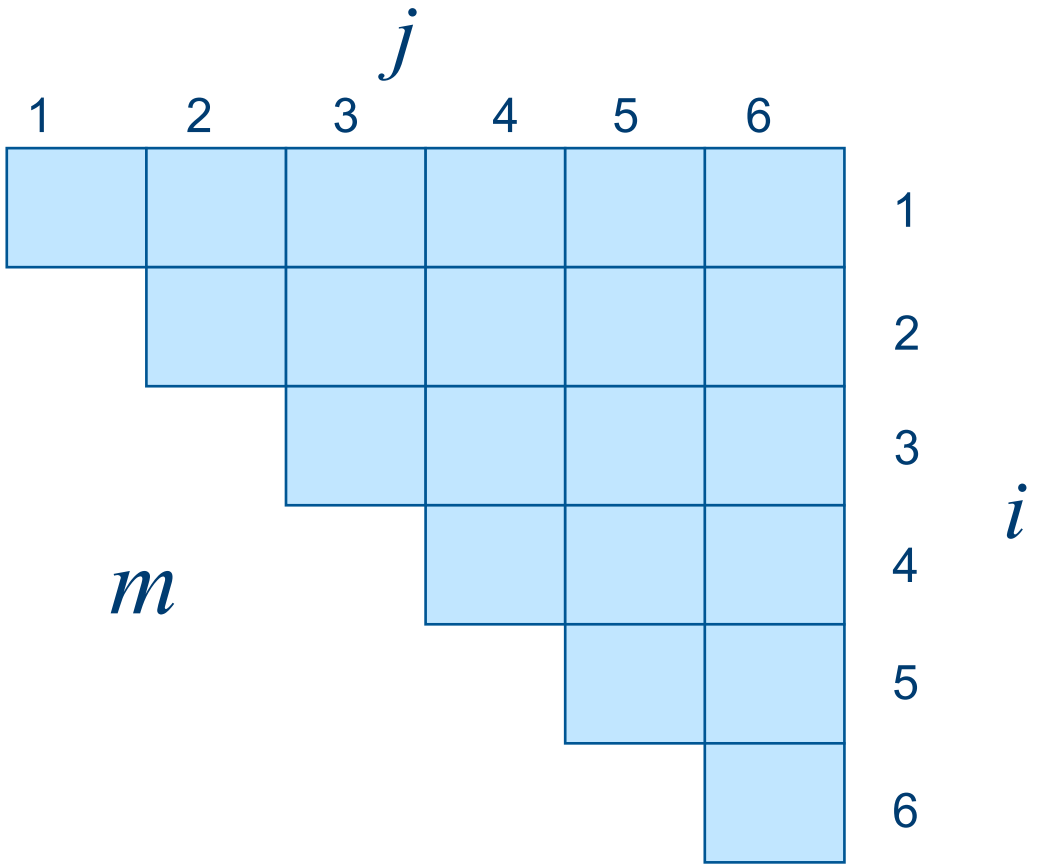
$$(A_i\, A_k)\,(A_{k+1} \,..\, A_j)$$

$$p_{i\text{-}1} \times p_k \qquad p_k \times p_j$$

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

Example

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25



Example

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

j

m

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

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

Example

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

j

1	2	3	4	5	6	
0	15,750	7,875	9,375	11,875	15,125	1
	0	2,625	4,375	7,125	10,500	2
		0	750	2,500	5,375	3
			0	1,000	3,500	4
				0	5,000	5
					0	6

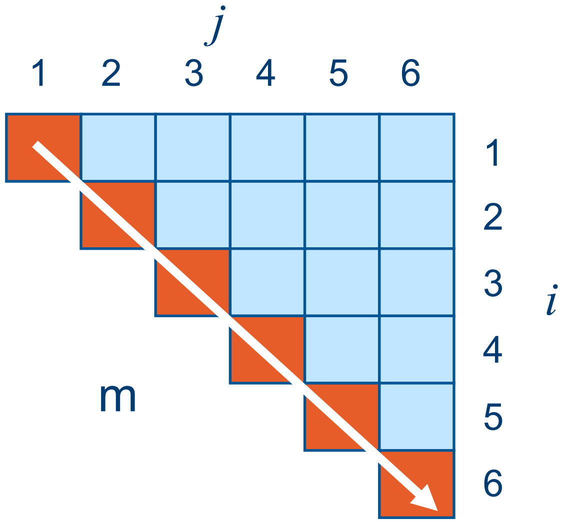
m

i

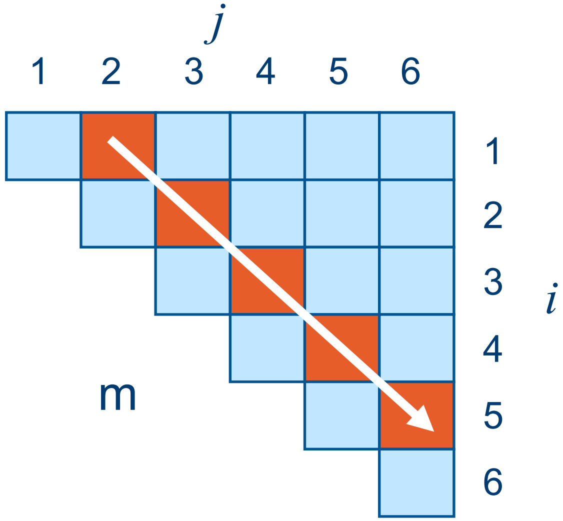
$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{if } i < j \end{cases}$$

$$\begin{aligned} m[2,5] &= \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 &= 0 + 2500 + 35 \cdot 15 \cdot 20 &= 13{,}000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 &= 2625 + 1000 + 35 \cdot 5 \cdot 20 &= 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 &= 4375 + 0 + 35 \cdot 10 \cdot 20 &= 11{,}375 \end{cases} \\ &= 7125\,. \end{aligned}$$

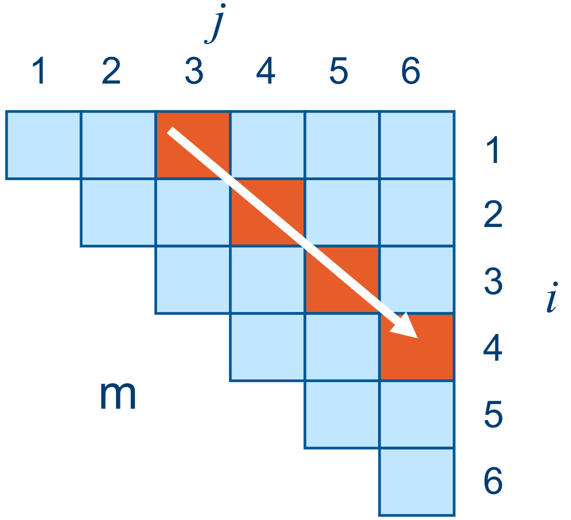
Example



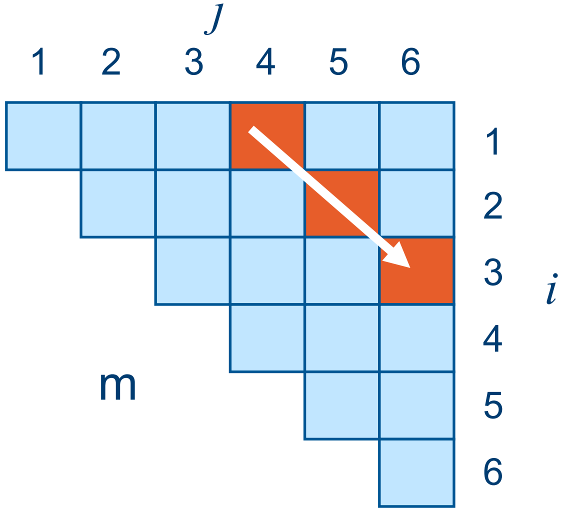
All parenthesizations of length 1



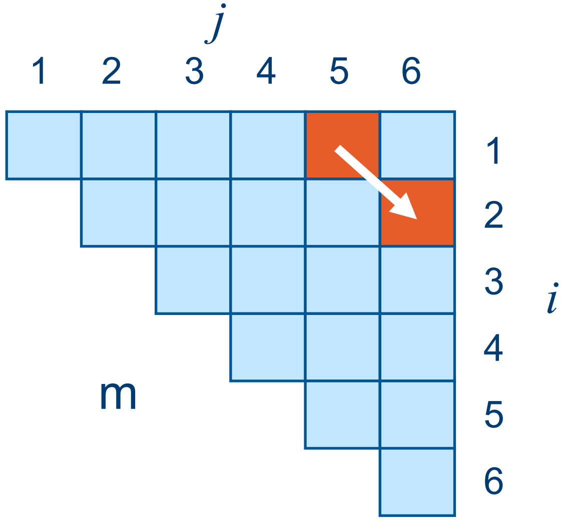
All parenthesizations of length 2



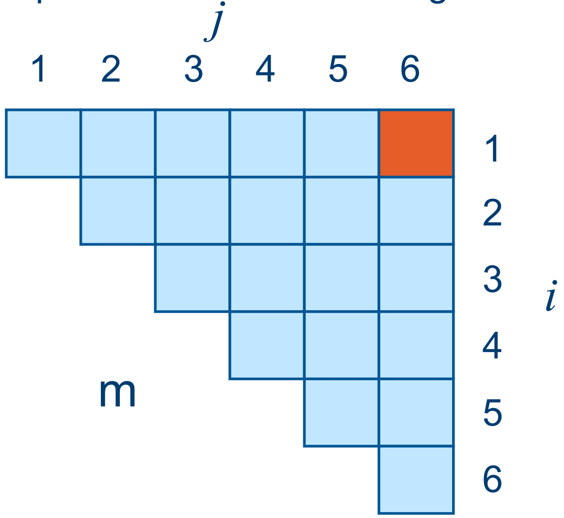
All parenthesizations of length 3



All parenthesizations of length 4



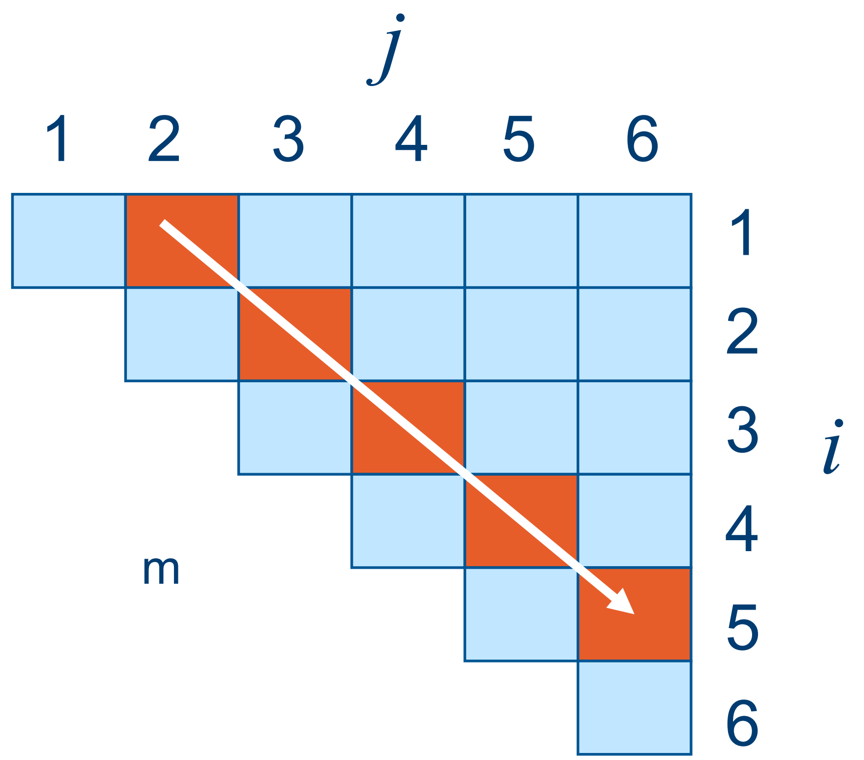
All parenthesizations of length 5



All parenthesizations of length 6

Matrix Chain Multiplication (DP Version)

```
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] = ∞
        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
```

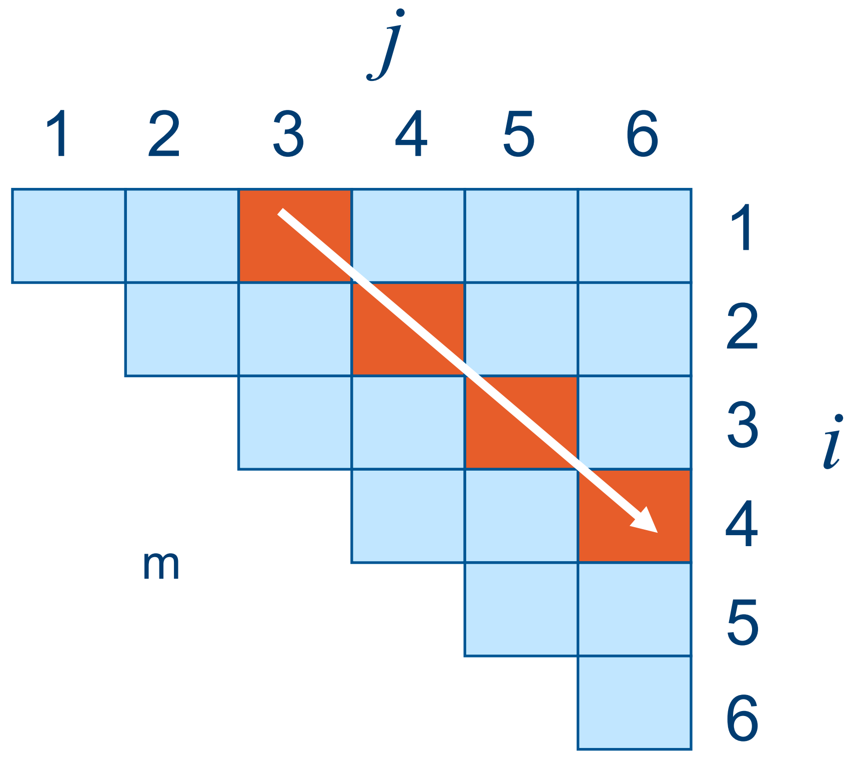


$l = 2$

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

Matrix Chain Multiplication (DP Version)

```
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] = ∞
        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
```

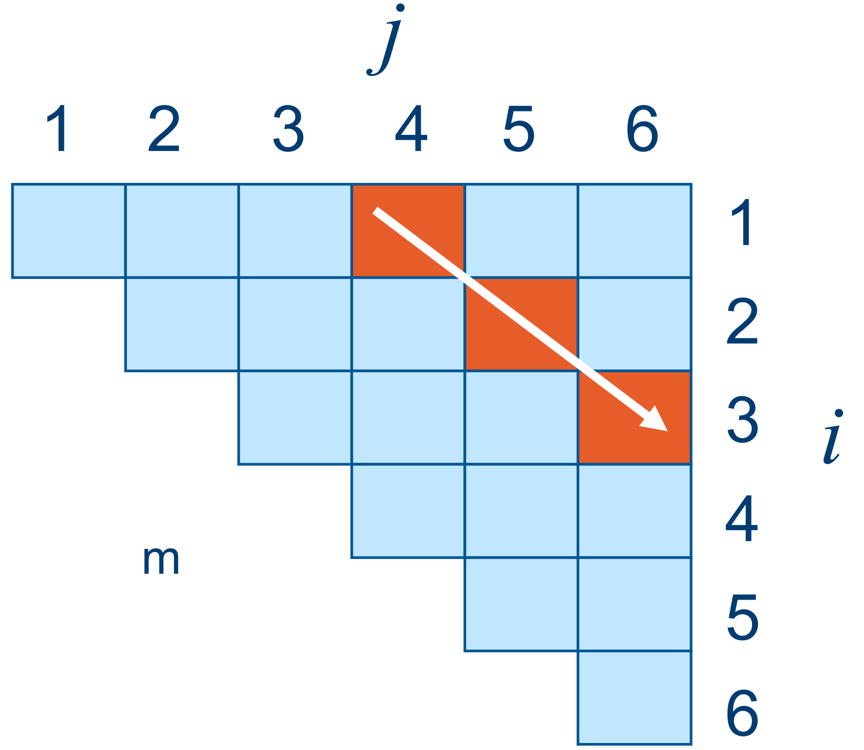


$l = 3$

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

Matrix Chain Multiplication (DP Version)

```
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] = ∞
        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
```

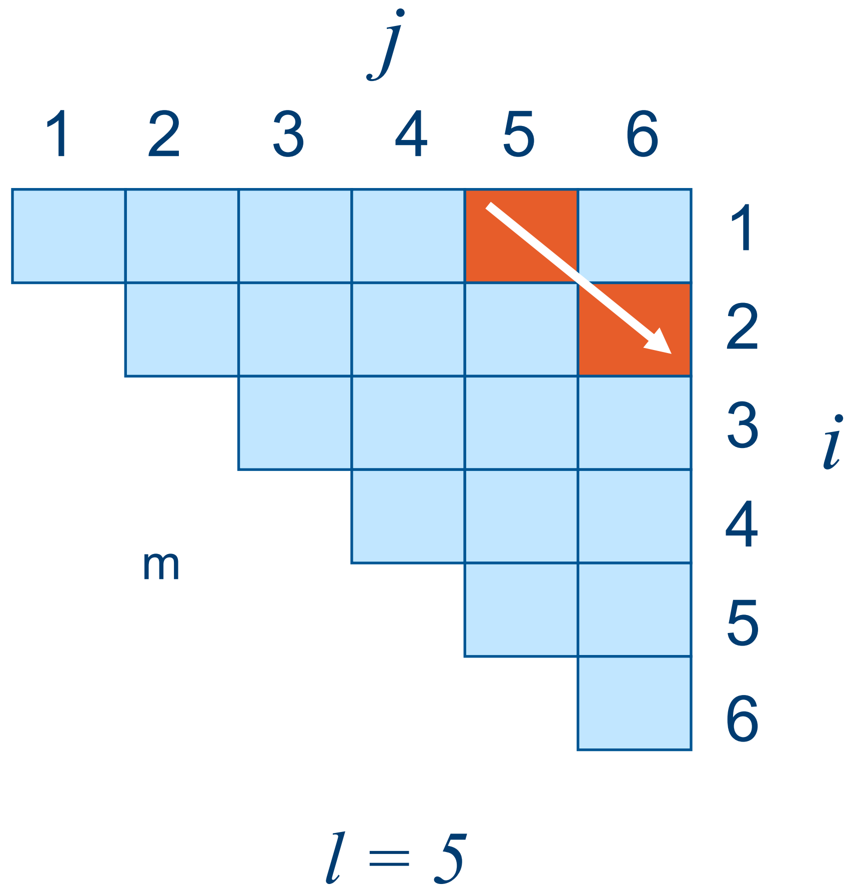


$l = 4$

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

Matrix Chain Multiplication (DP Version)

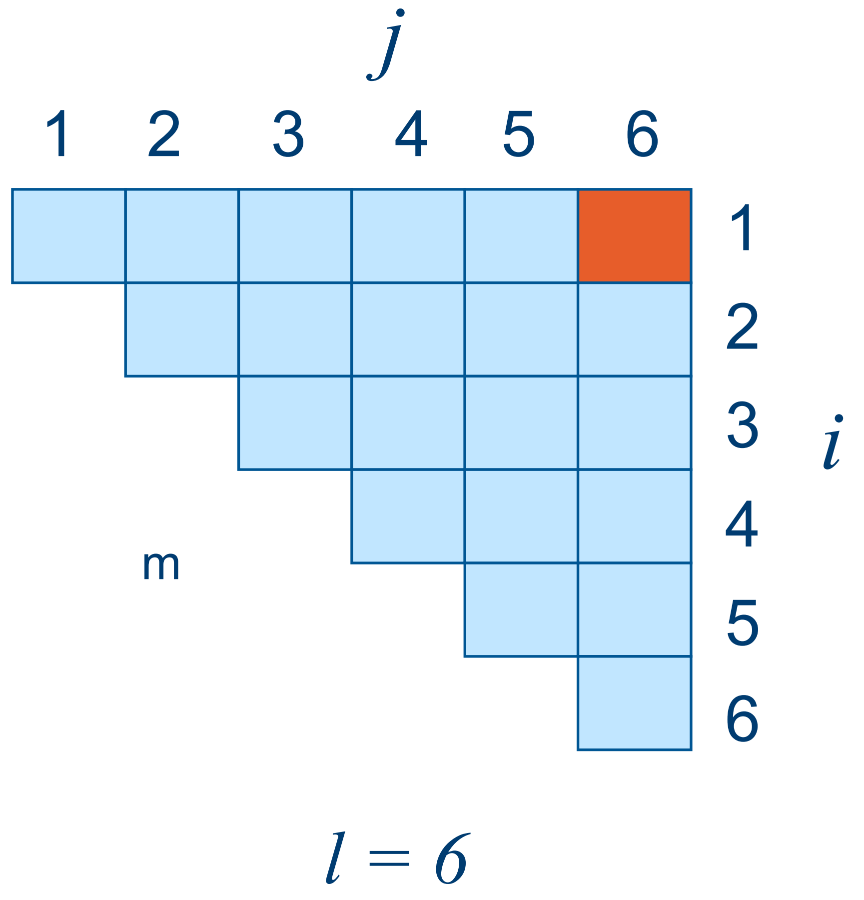
```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                    // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, 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
```



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

Matrix Chain Multiplication (DP Version)

```
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] = ∞
        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
```



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

Matrix Chain Multiplication (DP Version)

```
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] = ∞
        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
```

Time:

$$T(n) \in \theta(n^3)$$

Space:

$$S(n) \in \theta(n^2)$$

Matrix Chain Multiplication (D&C Version)

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
if i == j
    return 0
m[i, j] = ∞
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]
```

$$T(n) \in \Omega(2^n)$$

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

Matrix Chain Multiplication (D&C Version)

$$T(1) \geq 1,$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \qquad \text{for } n > 1$$

$$T(n) \geq 2\sum_{i=1}^{n-1} T(i) + n \qquad \text{(x)}$$

We shall prove that $T(n) = \Omega(2^n)$ using the substitution method.

or $T(n) \geq 2^{n-1}$ for all $n \geq 1$.

Base: $T(1) \geq 1 = 2^0$.

Induction:

(x) Induction Assumption

$$T(n) \geq 2\sum_{i=1}^{n-1} 2^{i-1} + n \qquad (T(i) \geq 2^{i-1}) \qquad i < n$$

Matrix Chain Multiplication (D&C Version)

$$T(n) \geq 2\sum_{i=1}^{n-1} T(i) + n$$

We shall prove that $T(n) = \Omega(2^n)$ using the substitution method.

or $T(n) \geq 2^{n-1}$ for all $n \geq 1$.

Base: $T(1) \geq 1 = 2^0$.

Induction:

$$\begin{aligned} T(n) &\geq 2\sum_{i=1}^{n-1} 2^{i-1} + n \qquad (T(i) \geq 2^{i-1}) \qquad i < n \\ &= 2\sum_{i=0}^{n-2} 2^i + n \\ &= 2(2^{n-1} - 1) + n \\ &= 2^n - 2 + n \\ &\geq 2^{n-1}, \end{aligned}$$

Wrap-up

- Dynamic programming lets you solve:
  - Any problem that has overlapping sub-problems
- Dynamic programming
  - Stores the optimal solutions to those sub-problems
  - Combines those sub-problem solutions to find the optimal solution for a larger sub-problem