Algorithms: Dynamic Programming (Matrix Chain Multi. and Longest Common Subsequence)

Ola Svensson

EPFL School of Computer and Communication Sciences

Lecture 11, 25.03.2025

DYNAMIC PROGRAMMING

(An algorithmic paradigm not a way of "programming")

DYNAMIC PROGRAMMING

(An algorithmic paradigm not a way of "programming")

What is $2^5 + 3 - \sqrt{16}$?

DYNAMIC PROGRAMMING

(An algorithmic paradigm not a way of "programming")

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

DYNAMIC PROGRAMMING

(An algorithmic paradigm not a way of "programming")

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

DYNAMIC PROGRAMMING

(An algorithmic paradigm not a way of "programming")

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

DYNAMIC PROGRAMMING

(An algorithmic paradigm not a way of "programming")

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

DYNAMIC PROGRAMMING

(An algorithmic paradigm not a way of "programming")

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

DYNAMIC PROGRAMMING

(An algorithmic paradigm not a way of "programming")

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

What is $2^5 + 3 - \sqrt{16}$?

Parenthesization	Cost computation	Cost
$A \times ((B \times C) \times D)$	$20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$	$120,200$
$(A \times (B \times C)) \times D$	$20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$	$60,200$
$(A \times B) \times (C \times D)$	$50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$	$7,000$

MATRIX-CHAIN MULTIPLICATION

Cost of Matrix Multiplication

$$A_{p,q} \times B_{q,r}$$

Cost of Matrix Multiplication

$$\underset{p}{\overset{q}{A_{p,q}}} \times \underset{q}{\overset{r}{B_{q,r}}}$$

$$\begin{matrix} p \\ \left\{ \right. \end{matrix} \overbrace{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,q) \\ (2,1) & (2,2) & \cdots & (2,q) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,q) \end{bmatrix}}^{q} \qquad \begin{matrix} q \\ \left\{ \right. \end{matrix} \overbrace{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (q,1) & (q,2) & \cdots & (q,r) \end{bmatrix}}^{r}$$

Cost of Matrix Multiplication

$$\underbrace{A_{p,q}}_{q} \times \underbrace{B_{q,r}}_{r}$$

$$p\left\{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,q) \\ (2,1) & (2,2) & \cdots & (2,q) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,q) \end{bmatrix}\right. \qquad q\left\{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (q,1) & (q,2) & \cdots & (q,r) \end{bmatrix}\right.$$

$$\Downarrow$$

$$\underbrace{C_{p,r}}_{r}$$

$$p\left\{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,r) \end{bmatrix}\right.$$

Cost of Matrix Multiplication

$$
\underbrace{A_{p,q}}_{q}
\begin{matrix} \\ p \end{matrix}
\begin{bmatrix}
(1,1) & (1,2) & \cdots & (1,q) \\
(2,1) & (2,2) & \cdots & (2,q) \\
\vdots & \vdots & \ddots & \vdots \\
(p,1) & (p,2) & \cdots & (p,q)
\end{bmatrix}
\times
\underbrace{B_{q,r}}_{r}
\begin{matrix} \\ q \end{matrix}
\begin{bmatrix}
(1,1) & (1,2) & \cdots & (1,r) \\
(2,1) & (2,2) & \cdots & (2,r) \\
\vdots & \vdots & \ddots & \vdots \\
(q,1) & (q,2) & \cdots & (q,r)
\end{bmatrix}
$$

$$
\Downarrow
$$

$$
\underbrace{C_{p,r}}_{r}
\begin{matrix} \\ p \end{matrix}
\begin{bmatrix}
(1,1) & (1,2) & \cdots & (1,r) \\
\boxed{(2,1)} & (2,2) & \cdots & (2,r) \\
\vdots & \vdots & \ddots & \vdots \\
(p,1) & (p,2) & \cdots & (p,r)
\end{bmatrix}
$$

Cost of Matrix Multiplication

$$
\underbrace{A_{p,q}}_{q}\ \times\ \underbrace{B_{q,r}}_{r}
$$

$$
p\left\{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,q) \\ (2,1) & (2,2) & \cdots & (2,q) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,q) \end{bmatrix}\right. \qquad q\left\{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (q,1) & (q,2) & \cdots & (q,r) \end{bmatrix}\right.
$$

$$
\Downarrow
$$

$$
\underbrace{C_{p,r}}_{r}
$$

$$
p\left\{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,r) \end{bmatrix}\right.
$$

Cost of Matrix Multiplication

$$
\underbrace{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,q) \\ (2,1) & (2,2) & \cdots & (2,q) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,q) \end{bmatrix}}_{q} \Big\}\, p \quad A_{p,q}
$$

$$
\times
$$

$$
\underbrace{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (q,1) & (q,2) & \cdots & (q,r) \end{bmatrix}}_{r} \Big\}\, q \quad B_{q,r}
$$

$$
\Downarrow
$$

$$
\underbrace{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,r) \end{bmatrix}}_{r} \Big\}\, p \quad C_{p,r}
$$

Cost of Matrix Multiplication

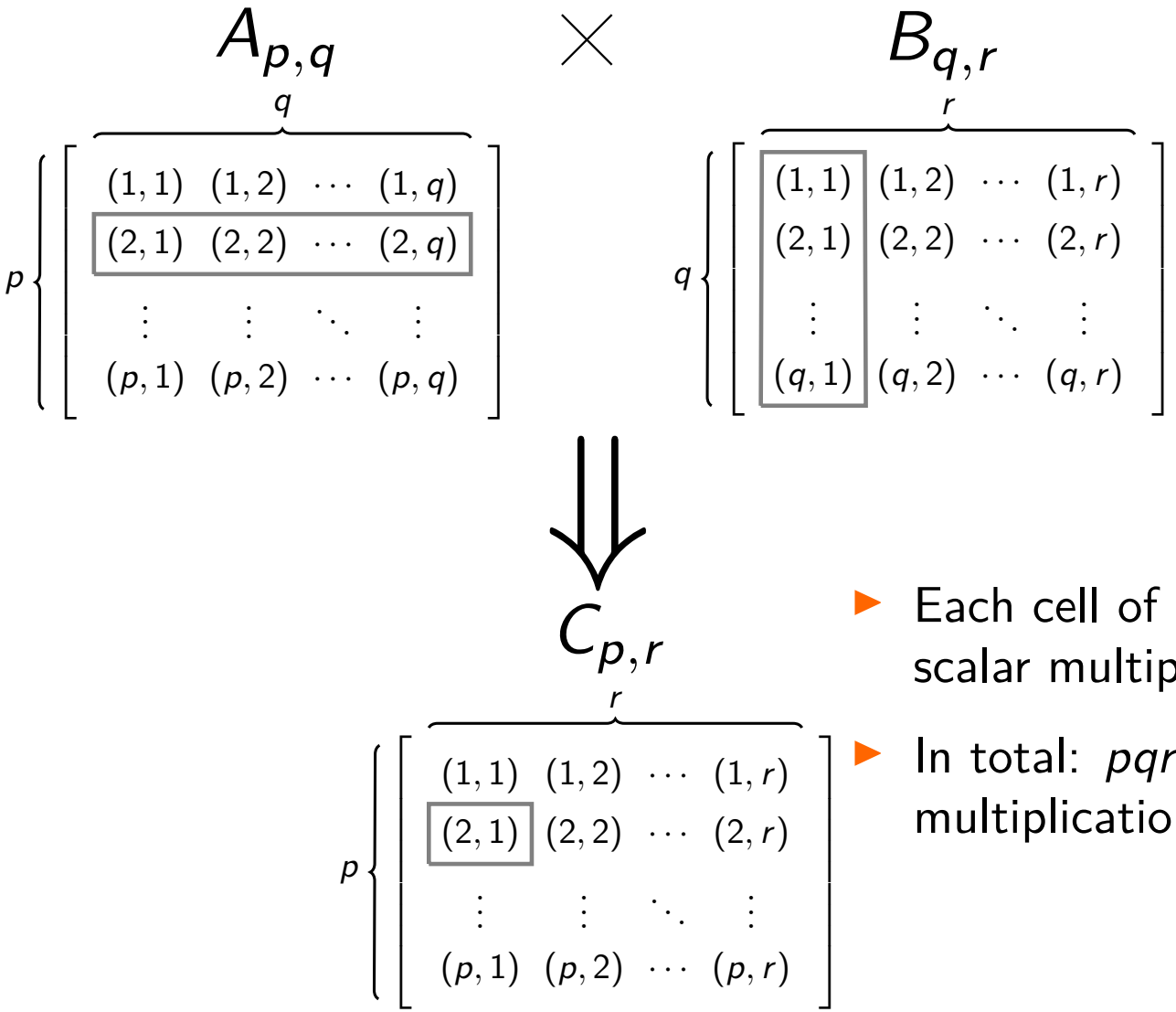
$$
\underbrace{A_{p,q}}_{q}\ p\left\{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,q) \\ (2,1) & (2,2) & \cdots & (2,q) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,q) \end{bmatrix}\right. \times \underbrace{B_{q,r}}_{r}\ q\left\{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (q,1) & (q,2) & \cdots & (q,r) \end{bmatrix}\right.
$$

$$
\Downarrow
$$

$$
\underbrace{C_{p,r}}_{r}\ p\left\{\begin{bmatrix} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,r) \end{bmatrix}\right.
$$

- Each cell of C requires q scalar multiplications.

Cost of Matrix Multiplication



- Each cell of C requires q scalar multiplications.
- In total: pqr scalar multiplications.

Cost of Matrix Multiplication

$$A_{p,q} \times B_{q,r} \Rightarrow C_{p,r}$$

$$A_{p,q} = \begin{array}{c} \overbrace{}^{q} \\ \left\{ p \right. \left[\begin{array}{cccc} (1,1) & (1,2) & \cdots & (1,q) \\ (2,1) & (2,2) & \cdots & (2,q) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,q) \end{array} \right] \end{array}$$

$$B_{q,r} = \begin{array}{c} \overbrace{}^{r} \\ \left\{ q \right. \left[\begin{array}{cccc} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (q,1) & (q,2) & \cdots & (q,r) \end{array} \right] \end{array}$$

$$C_{p,r} = \begin{array}{c} \overbrace{}^{r} \\ \left\{ p \right. \left[\begin{array}{cccc} (1,1) & (1,2) & \cdots & (1,r) \\ (2,1) & (2,2) & \cdots & (2,r) \\ \vdots & \vdots & \ddots & \vdots \\ (p,1) & (p,2) & \cdots & (p,r) \end{array} \right] \end{array}$$

- Each cell of C requires q scalar multiplications.
- In total: pqr scalar multiplications.
- The scalar multiplications dominate the time complexity.

Matrix Chain Multiplication

Definition

Input: A chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$.

Output: A full parenthesization of the product $A_1 A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.

Matrix Chain Multiplication

Definition

Input: A chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$.

Output: A full parenthesization of the product $A_1 A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.

Remarks

Matrix Chain Multiplication

Definition

Input: A chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$.

Output: A full parenthesization of the product $A_1 A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.

Remarks

- We are not asked to calculate the product, only find the best parenthesization.

Matrix Chain Multiplication

Definition

Input: A chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$.

Output: A full parenthesization of the product $A_1 A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.

Remarks

- We are not asked to calculate the product, only find the best parenthesization.
- The parenthesization can significantly affect the number of multiplications.

Matrix Chain Multiplication

Definition

Input: A chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$.

Output: A full parenthesization of the product $A_1 A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.

Example

Matrix Chain Multiplication

Definition

Input: A chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$.

Output: A full parenthesization of the product $A_1 A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.

Example

- A product $A_1 A_2 A_3$ with dimensions: 50×5, 5×100 and 100×10.

Matrix Chain Multiplication

Definition

Input: A chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$.

Output: A full parenthesization of the product $A_1 A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.

Example

- A product $A_1 A_2 A_3$ with dimensions: 50×5, 5×100 and 100×10.
- Calculating $(A_1 A_2) A_3$ requires: $50 \cdot 5 \cdot 100 + 50 \cdot 100 \cdot 10 = 75000$ scalar multiplications.

Matrix Chain Multiplication

Definition

Input: A chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$.

Output: A full parenthesization of the product $A_1 A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.

Example

- A product $A_1 A_2 A_3$ with dimensions: 50×5, 5×100 and 100×10.
- Calculating $(A_1 A_2) A_3$ requires: $50 \cdot 5 \cdot 100 + 50 \cdot 100 \cdot 10 = 75000$ scalar multiplications.
- Calculating $A_1 (A_2 A_3)$ requires: $5 \cdot 100 \cdot 10 + 50 \cdot 5 \cdot 10 = 7500$ scalar multiplications.

Optimal Substructure

Theorem

If:

- *the outermost parenthesization in an optimal solution is:* $(A_1 A_2 \cdots A_i)(A_{i+1} A_{i+2} \cdots A_n)$.
- P_L *and* P_R *are optimal parenthesizations for* $A_1 A_2 \cdots A_i$ *and* $A_{i+1} A_{i+2} \cdots A_n$, *respectively.*

Then, $((P_L) \cdot (P_R))$ *is an optimal parenthesizations for* $A_1 A_2 \cdots A_n$.

Optimal Substructure

Theorem

If:

- *the outermost parenthesization in an optimal solution is:* $(A_1 A_2 \cdots A_i)(A_{i+1} A_{i+2} \cdots A_n)$.
- P_L *and* P_R *are optimal parenthesizations for* $A_1 A_2 \cdots A_i$ *and* $A_{i+1} A_{i+2} \cdots A_n$, *respectively.*

Then, $((P_L) \cdot (P_R))$ *is an optimal parenthesizations for* $A_1 A_2 \cdots A_n$.

Proof

- Let $((O_L) \cdot (O_R))$ be an optimal parenthesization, where O_L and O_R are parenthesizations for $A_1 A_2 \cdots A_i$ and $A_{i+1} \cdots A_n$, respectively.

Optimal Substructure

Theorem

If:

- *the outermost parenthesization in an optimal solution is:* $(A_1 A_2 \cdots A_i)(A_{i+1} A_{i+2} \cdots A_n)$.
- P_L *and* P_R *are optimal parenthesizations for* $A_1 A_2 \cdots A_i$ *and* $A_{i+1} A_{i+2} \cdots A_n$, *respectively.*

Then, $((P_L) \cdot (P_R))$ *is an optimal parenthesizations for* $A_1 A_2 \cdots A_n$.

Proof

- Let $((O_L) \cdot (O_R))$ be an optimal parenthesization, where O_L and O_R are parenthesizations for $A_1 A_2 \cdots A_i$ and $A_{i+1} \cdots A_n$, respectively.
- Let $M(P)$ be the number of scalar multiplications required by a parenthesization.

Optimal Substructure

Theorem

If:

- *the outermost parenthesization in an optimal solution is:* $(A_1 A_2 \cdots A_i)(A_{i+1} A_{i+2} \cdots A_n)$.
- P_L *and* P_R *are optimal parenthesizations for* $A_1 A_2 \cdots A_i$ *and* $A_{i+1} A_{i+2} \cdots A_n$, *respectively.*

Then, $((P_L) \cdot (P_R))$ *is an optimal parenthesizations for* $A_1 A_2 \cdots A_n$.

Proof

$$M((O_L) \cdot (O_R))$$

Optimal Substructure

Theorem

If:

- *the outermost parenthesization in an optimal solution is:* $(A_1 A_2 \cdots A_i)(A_{i+1} A_{i+2} \cdots A_n)$.
- P_L *and* P_R *are optimal parenthesizations for* $A_1 A_2 \cdots A_i$ *and* $A_{i+1} A_{i+2} \cdots A_n$, *respectively.*

Then, $((P_L) \cdot (P_R))$ *is an optimal parenthesizations for* $A_1 A_2 \cdots A_n$.

Proof

$$M((O_L) \cdot (O_R)) = p_0 \cdot p_i \cdot p_n + M(O_L) + M(O_R)$$

Optimal Substructure

Theorem

If:

- *the outermost parenthesization in an optimal solution is:* $(A_1 A_2 \cdots A_i)(A_{i+1} A_{i+2} \cdots A_n)$.
- P_L *and* P_R *are optimal parenthesizations for* $A_1 A_2 \cdots A_i$ *and* $A_{i+1} A_{i+2} \cdots A_n$, *respectively.*

Then, $((P_L) \cdot (P_R))$ *is an optimal parenthesizations for* $A_1 A_2 \cdots A_n$.

Proof

$$M((O_L) \cdot (O_R)) = p_0 \cdot p_i \cdot p_n + M(O_L) + M(O_R)$$

- Since P_L and P_R are optimal: $M(P_L) \leq M(O_L)$ and $M(P_R) \leq M(O_R)$.

Optimal Substructure

Theorem

If:

- *the outermost parenthesization in an optimal solution is:* $(A_1 A_2 \cdots A_i)(A_{i+1} A_{i+2} \cdots A_n)$.
- P_L *and* P_R *are optimal parenthesizations for* $A_1 A_2 \cdots A_i$ *and* $A_{i+1} A_{i+2} \cdots A_n$, *respectively.*

Then, $((P_L) \cdot (P_R))$ *is an optimal parenthesizations for* $A_1 A_2 \cdots A_n$.

Proof

$$\begin{aligned} M((O_L) \cdot (O_R)) &= p_0 \cdot p_i \cdot p_n + M(O_L) + M(O_R) \\ &\geq p_0 \cdot p_i \cdot p_n + M(P_L) + M(P_R) \end{aligned}$$

- Since P_L and P_R are optimal: $M(P_L) \leq M(O_L)$ and $M(P_R) \leq M(O_R)$.

Optimal Substructure

Theorem

If:

- *the outermost parenthesization in an optimal solution is:* $(A_1 A_2 \cdots A_i)(A_{i+1} A_{i+2} \cdots A_n)$.
- P_L *and* P_R *are optimal parenthesizations for* $A_1 A_2 \cdots A_i$ *and* $A_{i+1} A_{i+2} \cdots A_n$, *respectively.*

Then, $((P_L) \cdot (P_R))$ *is an optimal parenthesizations for* $A_1 A_2 \cdots A_n$.

Proof

$$\begin{aligned} M((O_L) \cdot (O_R)) &= p_0 \cdot p_i \cdot p_n + M(O_L) + M(O_R) \\ &\geq p_0 \cdot p_i \cdot p_n + M(P_L) + M(P_R) = M((P_L) \cdot (P_R)) \ . \end{aligned}$$

- Since P_L and P_R are optimal: $M(P_L) \leq M(O_L)$ and $M(P_R) \leq M(O_R)$.

Recursive Formula

- Let $m[i, j]$ be the optimal number of scalar multiplications for calculating $A_i A_{i+1} \cdots, A_j$.

Recursive Formula

- Let $m[i,j]$ be the optimal number of scalar multiplications for calculating $A_i A_{i+1} \cdots, A_j$.

- $m[i,j]$ can be expressed recursively as follows:

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

Recursive Formula

- Let $m[i,j]$ be the optimal number of scalar multiplications for calculating $A_i A_{i+1} \cdots, A_j$.

- $m[i,j]$ can be expressed recursively as follows:

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

- Each $m[i,j]$ depend only on subproblems with smaller $j - i$.

Recursive Formula

- Let $m[i,j]$ be the optimal number of scalar multiplications for calculating $A_i A_{i+1} \cdots, A_j$.
- $m[i,j]$ can be expressed recursively as follows:

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

- Each $m[i,j]$ depend only on subproblems with smaller $j - i$.
- A bottom-up algorithm should solve subproblems in increasing $j - i$ order.

Example

Instance

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

Bottom-Up Algorithm

```
MATRIX-CHAIN-ORDER(p)
1   n = p.length − 1
2   let m[1..n, 1..n] and s[1..n, 1..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] = ∞
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
```

Bottom-Up Algorithm

```
MATRIX-CHAIN-ORDER(p)
1   n = p.length − 1
2   let m[1..n, 1..n] and s[1..n, 1..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] = ∞
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
```

⇐ s stores the optimal choice

Example

Instance

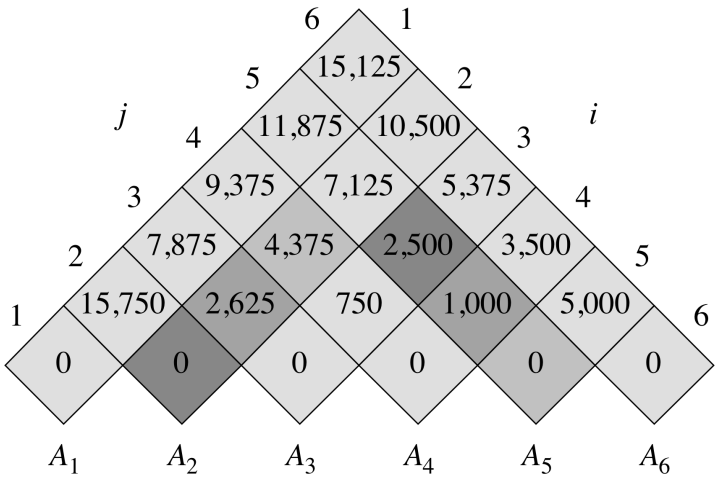
matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimensions	30×35	35×15	15×5	5×10	10×20	20×25

Example

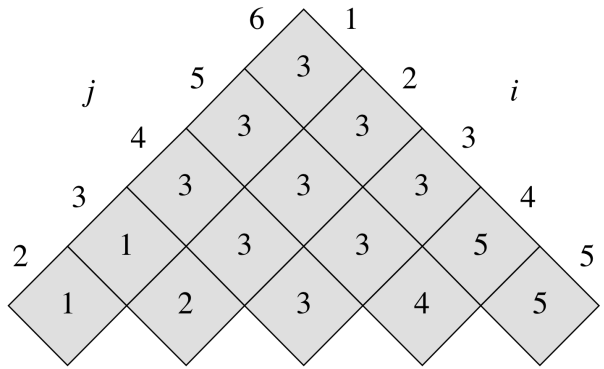
Instance

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

m



s

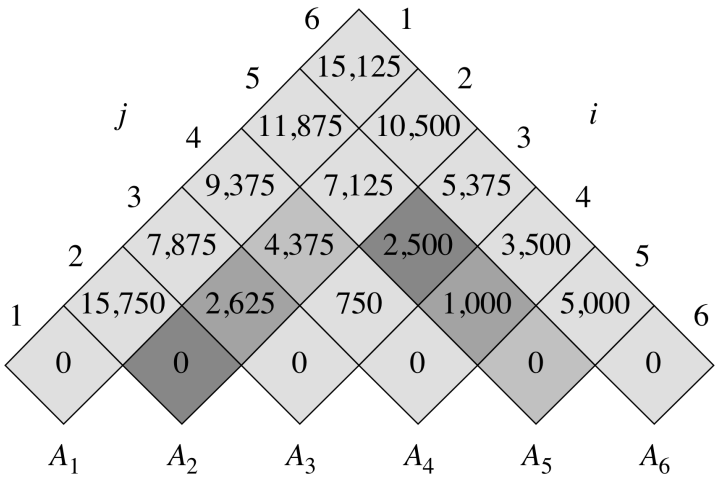


Example

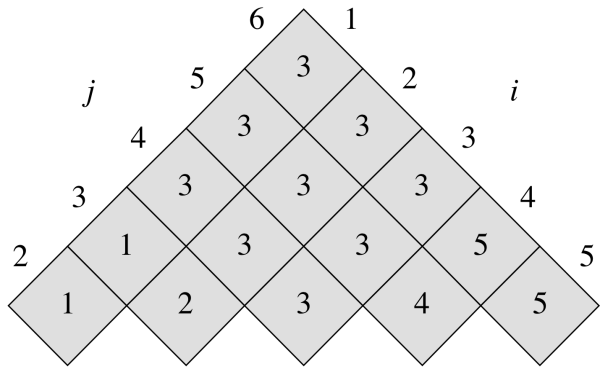
Instance

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

m



s

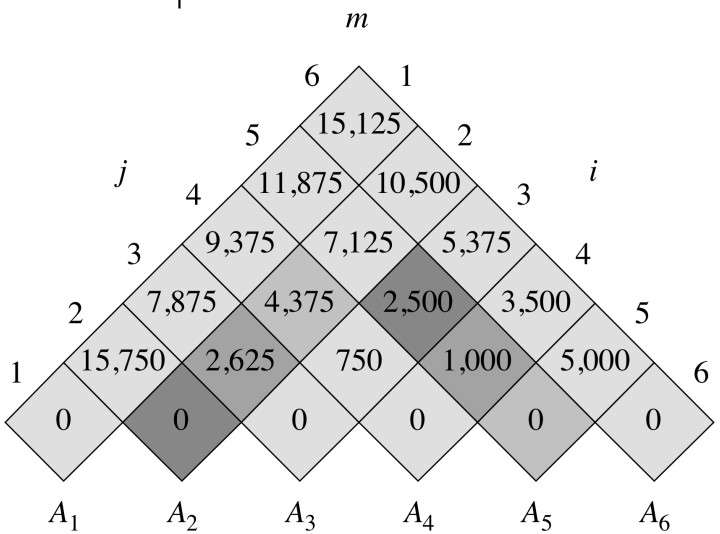


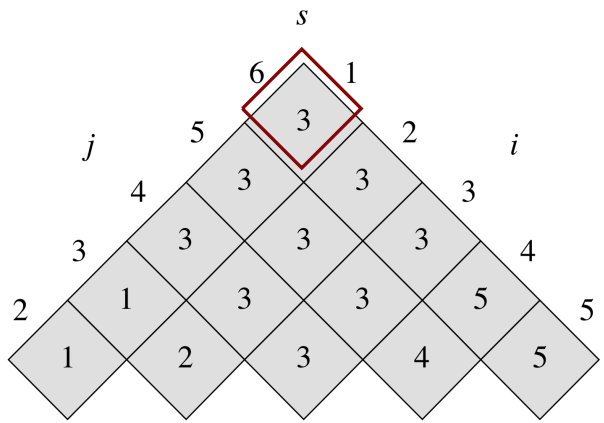
$A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5 \quad A_6$

Example

Instance

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



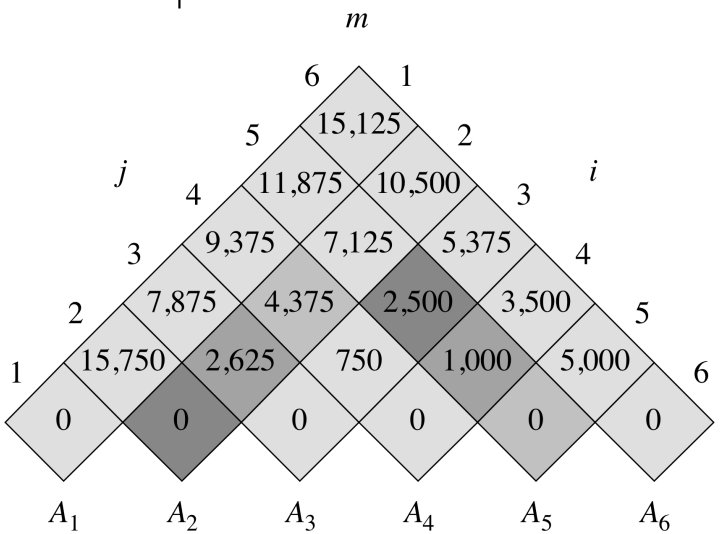


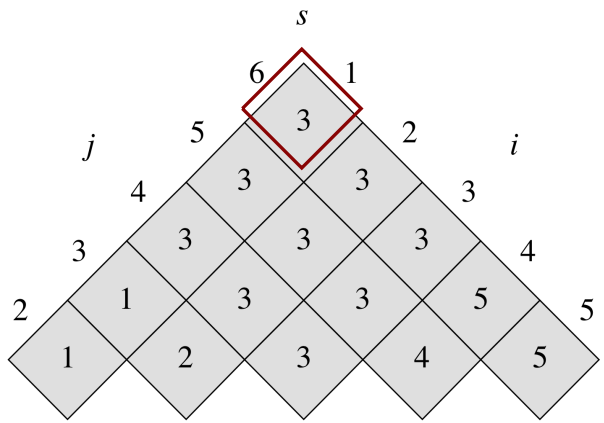
$A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5 \quad A_6$

Example

Instance

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



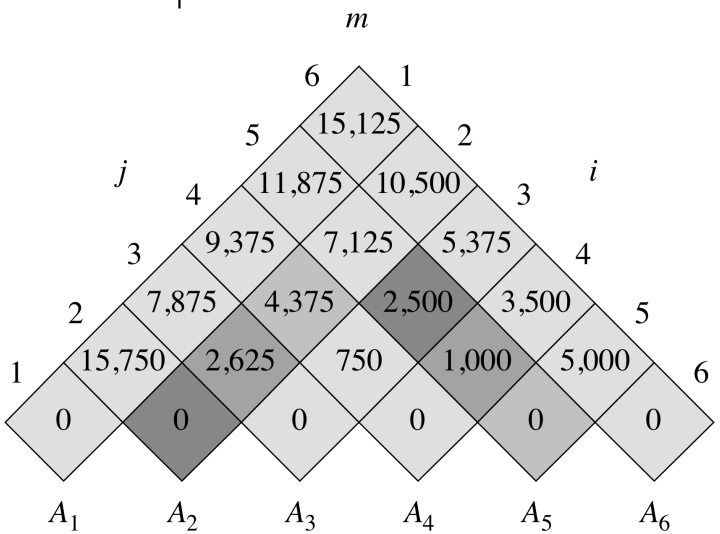


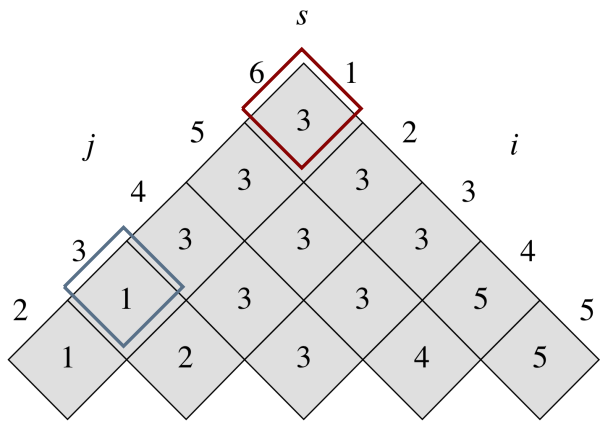
$$(A_1 \quad A_2 \quad A_3 \,)(\, A_4 \quad A_5 \quad A_6)$$

Example

Instance

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



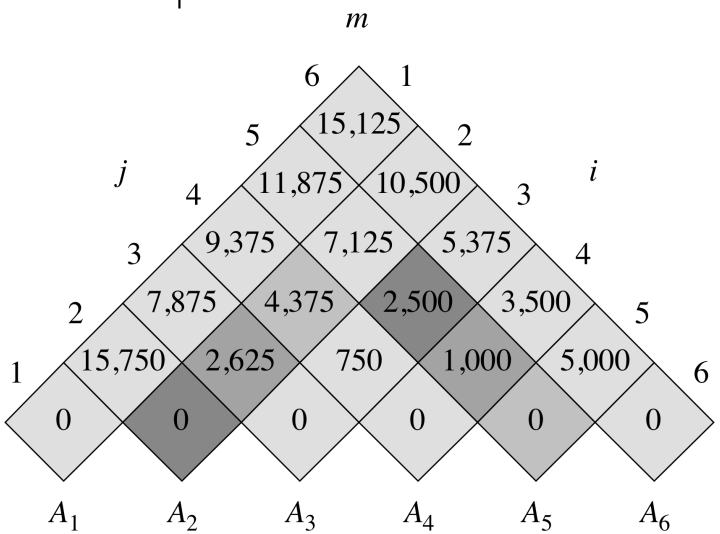


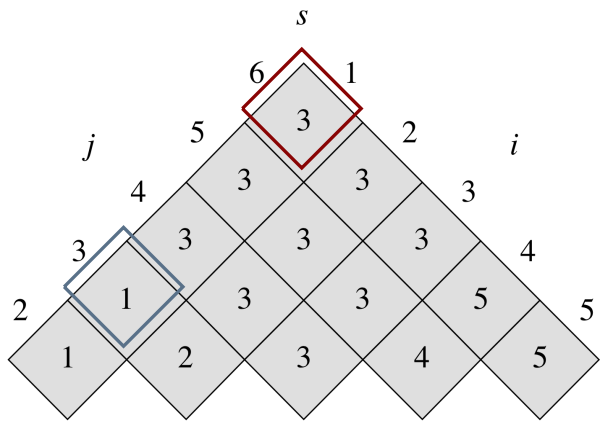
$$(A_1 \quad A_2 \quad A_3 \,)(\, A_4 \quad A_5 \quad A_6)$$

Example

Instance

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



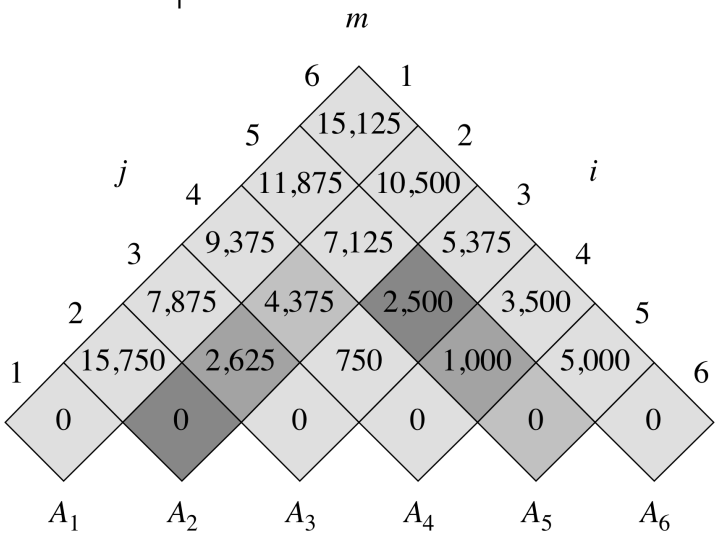


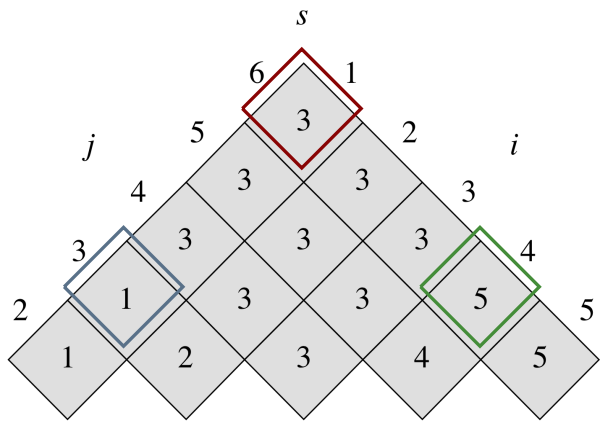
$$(A_1 \quad (A_2 \quad A_3))(\ A_4 \quad A_5 \quad A_6)$$

Example

Instance

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



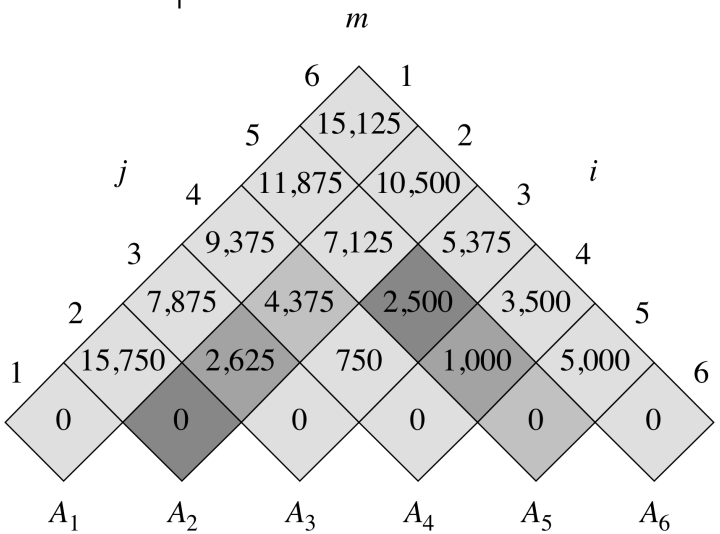


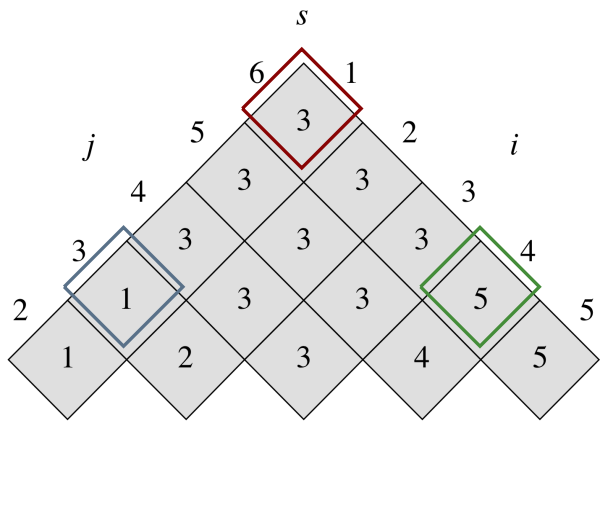
$$(A_1 \quad (A_2 \quad A_3))(\ A_4 \quad A_5 \quad A_6)$$

Example

Instance

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





$$(A_1 \quad (A_2 \quad A_3))((A_4 \quad A_5) \quad A_6)$$

Algorithm for Recovering an Optimal Solution

```
PRINT-OPTIMAL-PARENS(s, i, j)
1   if i == j
2       print "A_i"
3   else print "("
4       PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5       PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6       print ")"
```

Summary

Choice:

Summary

Choice: where to make the outermost parenthesis

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

Summary

Choice: where to make the outermost parenthesis

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

Optimal substructure:

Summary

Choice: where to make the outermost parenthesis

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

Optimal substructure: to obtain an optimal solution, we need to parenthesize the two remaining expressions in an optimal way

Summary

Choice: where to make the outermost parenthesis

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

Optimal substructure: to obtain an optimal solution, we need to parenthesize the two remaining expressions in an optimal way

Hence, if we let $m[i, j]$ be the optimal value for chain multiplication of matrices $A_i, \ldots, A_j$, we can express $m[i, j]$ recursively as follows

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

Summary

Choice: where to make the outermost parenthesis

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

Optimal substructure: to obtain an optimal solution, we need to parenthesize the two remaining expressions in an optimal way

Hence, if we let $m[i,j]$ be the optimal value for chain multiplication of matrices $A_i, \ldots, A_j$, we can express $m[i,j]$ recursively as follows

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

Overlapping subproblem: Solve recurrence using top-down with memoization or bottom-up which yields an algorithm that runs in time $\Theta(n^3)$.

LONGEST COMMON SUBSEQUENCE

Longest common subsequence

Definition

INPUT: 2 sequences, $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$.

OUTPUT: A subsequence common to both whose length is longest.

A subsequence doesn't have to be consecutive, but it has to be in order

Longest common subsequence

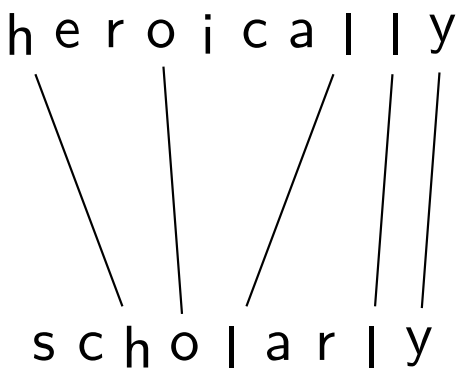
Definition

INPUT: 2 sequences, $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$.

OUTPUT: A subsequence common to both whose length is longest.

A subsequence doesn't have to be consecutive, but it has to be in order

Example:



First ideas fail

Brute force: For every subsequence of X, check whether it's a subsequence of Y

First ideas fail

Brute force: For every subsequence of X, check whether it's a subsequence of Y

Time: $\Theta(n2^m)$

- 2^m subsequences of X to check
- Each subsequence takes $\Theta(n)$ time to check: scan Y for first letter, from there scan for second, and so on

First ideas fail

Brute force: For every subsequence of X, check whether it's a subsequence of Y

Time: $\Theta(n2^m)$

- 2^m subsequences of X to check
- Each subsequence takes $\Theta(n)$ time to check: scan Y for first letter, from there scan for second, and so on

No natural greedy algorithm for the problem :(

Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

B A B D B A

D A C B C B A

Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

Dynamic programming comes to the rescue

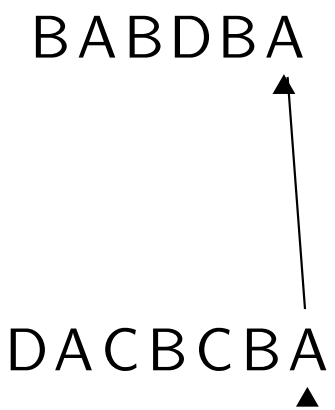
Start at the end of both words and move to the left step-by-step

Choice?

Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

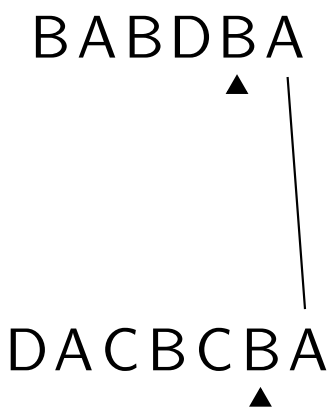
Choice? If the same, pick letter to be in the subsequence



Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

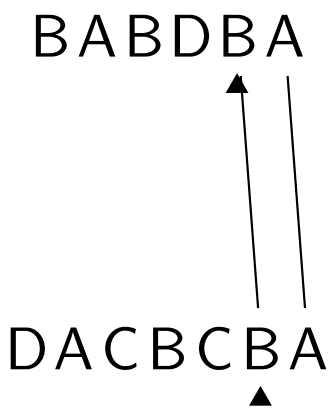
Choice? If the same, pick letter to be in the subsequence



Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

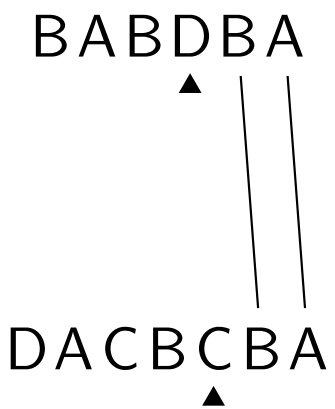
Choice? If the same, pick letter to be in the subsequence



Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

Choice? If the same, pick letter to be in the subsequence

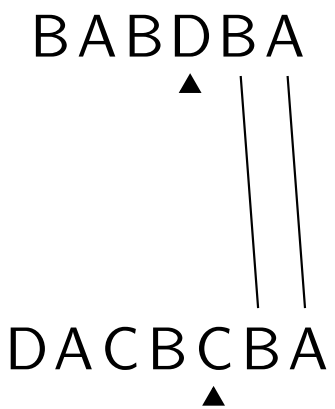


Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

Choice? If the same, pick letter to be in the subsequence

If not the same, optimal subsequence can be obtained by moving a step to the left in one of the words

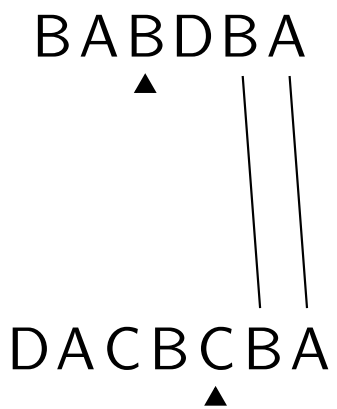


Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

Choice? If the same, pick letter to be in the subsequence

If not the same, optimal subsequence can be obtained by moving a step to the left in one of the words

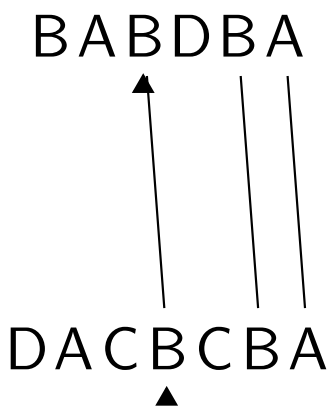


Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

Choice? If the same, pick letter to be in the subsequence

If not the same, optimal subsequence can be obtained by moving a step to the left in one of the words

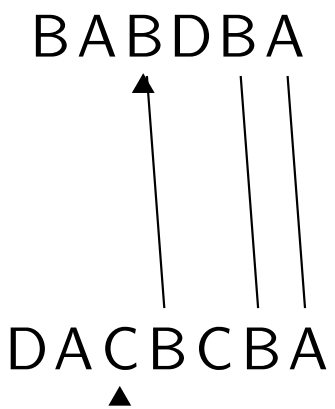


Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

Choice? If the same, pick letter to be in the subsequence

If not the same, optimal subsequence can be obtained by moving a step to the left in one of the words

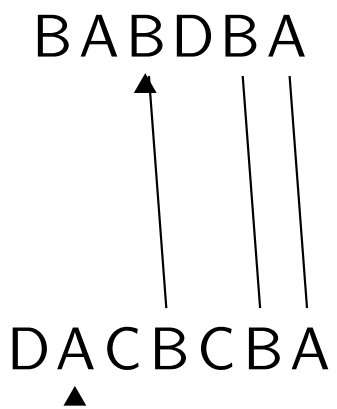


Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

Choice? If the same, pick letter to be in the subsequence

If not the same, optimal subsequence can be obtained by moving a step to the left in one of the words

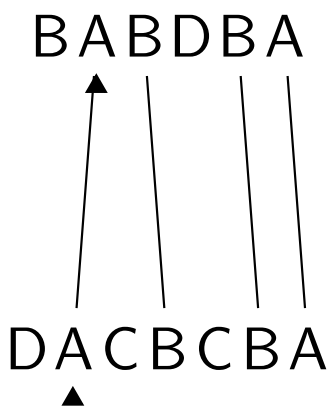


Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

Choice? If the same, pick letter to be in the subsequence

If not the same, optimal subsequence can be obtained by moving a step to the left in one of the words

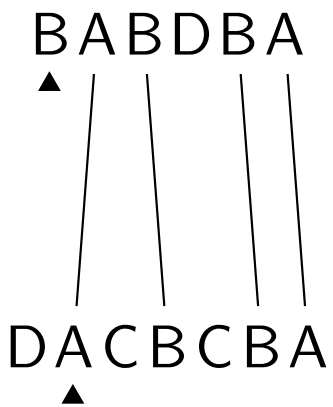


Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

Choice? If the same, pick letter to be in the subsequence

If not the same, optimal subsequence can be obtained by moving a step to the left in one of the words

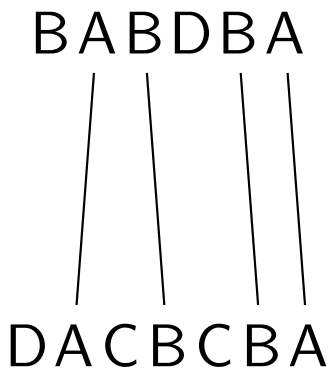


Dynamic programming comes to the rescue

Start at the end of both words and move to the left step-by-step

Choice? If the same, pick letter to be in the subsequence

If not the same, optimal subsequence can be obtained by moving a step to the left in one of the words



Optimal substructure

Let X_i and Y_j denote the prefixes $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots y_j \rangle$

Optimal substructure

Let X_i and Y_j denote the prefixes $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots y_j \rangle$

Theorem

Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X_i and Y_j.

1. *If $x_i = y_j$ then $z_k = x_i = y_j$ and Z_{k-1} is an LCS of X_{i-1} and Y_{j-1}*

Optimal substructure

Let X_i and Y_j denote the prefixes $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots y_j \rangle$

Theorem

Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X_i and Y_j.

1. *If $x_i = y_j$ then $z_k = x_i = y_j$ and Z_{k-1} is an LCS of X_{i-1} and Y_{j-1}*

Proof. Suppose $z_k \neq x_i = y_j$

$$x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_{i-1} \quad x_i$$

$$y_1 \quad y_2 \quad \cdots \quad y_{j-1} \quad y_j$$

Optimal substructure

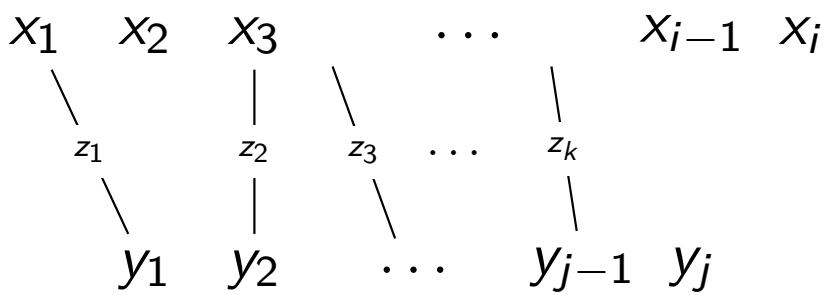
Let X_i and Y_j denote the prefixes $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots y_j \rangle$

Theorem

Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X_i and Y_j.

1. *If $x_i = y_j$ then $z_k = x_i = y_j$ and Z_{k-1} is an LCS of X_{i-1} and Y_{j-1}*

Proof. Suppose $z_k \neq x_i = y_j$



Optimal substructure

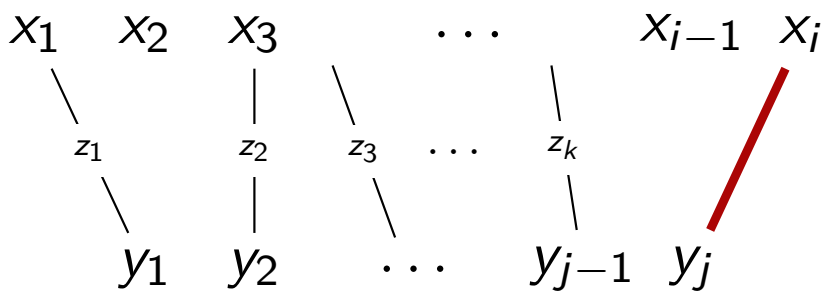
Let X_i and Y_j denote the prefixes $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots y_j \rangle$

Theorem

Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X_i and Y_j.

1. *If $x_i = y_j$ then $z_k = x_i = y_j$ and Z_{k-1} is an LCS of X_{i-1} and Y_{j-1}*

Proof. Suppose $z_k \neq x_i = y_j$ but then $Z' = \langle z_1, \ldots, z_k, x_i \rangle$ is a common subsequence of X_i and Y_j which contradicts Z being a LCS.



Optimal substructure

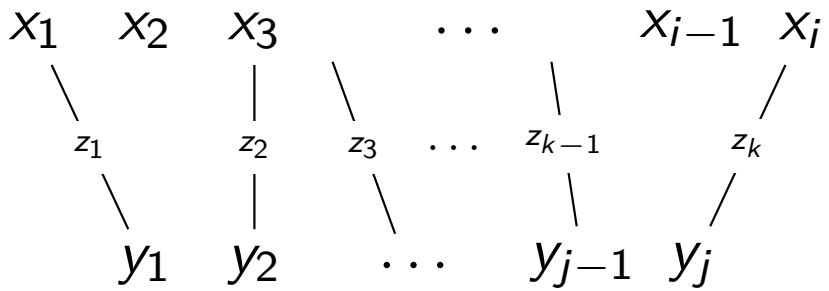
Let X_i and Y_j denote the prefixes $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots y_j \rangle$

Theorem

Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X_i and Y_j.

1. *If $x_i = y_j$ then $z_k = x_i = y_j$ and Z_{k-1} is an LCS of X_{i-1} and Y_{j-1}*

Proof. Similarly suppose that Z_{k-1} is not a LCS of X_{i-1} and Y_{j-1}



Optimal substructure

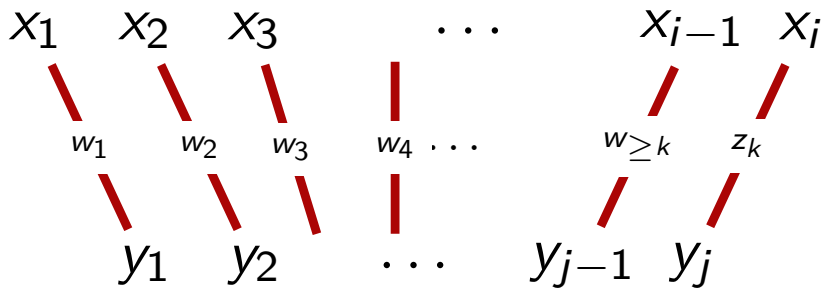
Let X_i and Y_j denote the prefixes $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots y_j \rangle$

Theorem

Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X_i and Y_j.

1. *If $x_i = y_j$ then $z_k = x_i = y_j$ and Z_{k-1} is an LCS of X_{i-1} and Y_{j-1}*

Proof. Similarly suppose that Z_{k-1} is not a LCS of X_{i-1} and Y_{j-1} but then exists a common subsequence W of X_{i-1} and Y_{j-1} that has length $\geq k$ which in turn implies that $\langle W, z_k \rangle$ has length $\geq k + 1$ contradicting the optimality of Z



Optimal substructure

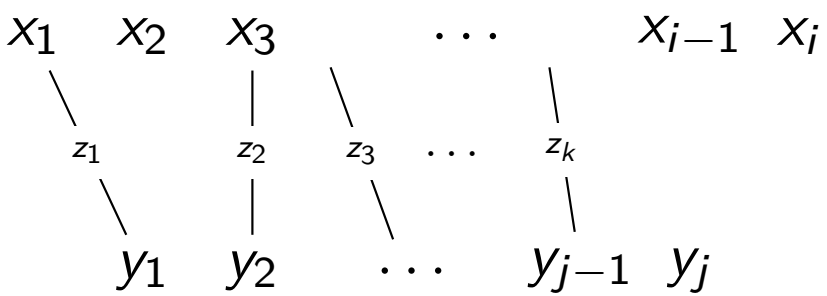
Let X_i and Y_j denote the prefixes $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots y_j \rangle$

Theorem

Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X_i and Y_j.

1. *If $x_i = y_j$ then $z_k = x_i = y_j$ and Z_{k-1} is an LCS of X_{i-1} and Y_{j-1}*
2. *If $x_i \neq y_j$, then $z_k \neq x_i \Rightarrow Z$ is an LCS of X_{i-1} and Y_j*

Proof. Z is a common subsequence to X_{i-1} and Y_j. Suppose Z is not a LCS to X_{i-1} and Y_j



Optimal substructure

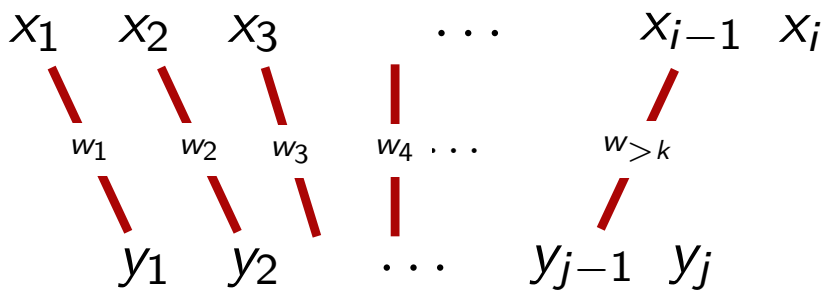
Let X_i and Y_j denote the prefixes $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots y_j \rangle$

Theorem

Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X_i and Y_j.

1. *If $x_i = y_j$ then $z_k = x_i = y_j$ and Z_{k-1} is an LCS of X_{i-1} and Y_{j-1}*
2. *If $x_i \neq y_j$, then $z_k \neq x_i \Rightarrow Z$ is an LCS of X_{i-1} and Y_j*

Proof. Z is a common subsequence to X_{i-1} and Y_j. Suppose Z is not a LCS to X_{i-1} and Y_j but then exists a common subsequence W of X_{i-1} and Y_j that has length $> k$ and, as it is also a common subsequence to X_i and Y_j, it contradicts the optimality of Z



Optimal substructure

Let X_i and Y_j denote the prefixes $\langle x_1, x_2, \ldots, x_i \rangle$ and $\langle y_1, y_2, \ldots y_j \rangle$

Theorem

Let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X_i and Y_j.

1. *If $x_i = y_j$ then $z_k = x_i = y_j$ and Z_{k-1} is an LCS of X_{i-1} and Y_{j-1}*
2. *If $x_i \neq y_j$, then $z_k \neq x_i \Rightarrow Z$ is an LCS of X_{i-1} and Y_j*
3. *If $x_i \neq y_j$, then $z_k \neq y_j \Rightarrow Z$ is an LCS of X_i and Y_{j-1}*

Proof. Same argument as for (2).

From the above theorem, we know that the length of a LCS of X_i, Y_j is

$$1 + \text{LCS of } X_{i-1} \text{ and } Y_{j-1} \qquad \text{if } x_i = y_j$$

$$\text{either LCS of } X_{i-1}, Y_j \text{ or LCS of } X_i, Y_{j-1} \qquad \text{otherwise}$$

Recursive formulation

Define $c[i, j] =$ length of LCS of X_i and Y_j. We want $c[m, n]$

$$c[i, j] = \begin{cases}$$

Recursive formulation

Define $c[i, j] =$ length of LCS of X_i and Y_j. We want $c[m, n]$

$$c[i, j] = \begin{cases} & \text{if } i = 0 \text{ or } j = 0 \end{cases}$$

Recursive formulation

Define $c[i, j] =$ length of LCS of X_i and Y_j. We want $c[m, n]$

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \end{cases}$$

Recursive formulation

Define $c[i, j] =$ length of LCS of X_i and Y_j. We want $c[m, n]$

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ & \text{if } i, j > 0 \text{ and } x_i = y_j \end{cases}$$

Recursive formulation

Define $c[i, j] =$ length of LCS of X_i and Y_j. We want $c[m, n]$

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

Recursive formulation

Define $c[i, j] =$ length of LCS of X_i and Y_j. We want $c[m, n]$

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

Recursive formulation

Define $c[i, j] =$ length of LCS of X_i and Y_j. We want $c[m, n]$

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

Recursive formulation

Define $c[i,j]$ = length of LCS of X_i and Y_j. We want $c[m,n]$

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

- Naive implementation solves same problems many many times

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0							
2	0							
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0							
2	0							
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0						
2	0							
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0					
2	0							
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0				
2	0							
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1			
2	0							
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1		
2	0							
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	
2	0							
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0							
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0							
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0						
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1					
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1				
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1			
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1		
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0							
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0						
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1					
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1				
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2			
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2		
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0							
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1						
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1					
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1				
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2			
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2		
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0							
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1						
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1					
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1				
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2			
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2		
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	3
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	3
6	0							

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	3
6	0	1						

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	3
6	0	1	2					

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	3
6	0	1	2	2				

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	3
6	0	1	2	2	2			

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	3
6	0	1	2	2	2	2		

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	3
6	0	1	2	2	2	2	3	

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	3
6	0	1	2	2	2	2	3	4

Bottom-up approach for LCS

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1
2	0	0	1	1	1	1	1	2
3	0	0	1	1	2	2	2	2
4	0	1	1	1	2	2	2	2
5	0	1	1	1	2	2	3	3
6	0	1	2	2	2	2	3	4

Longest common subsequence has length 4

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0							
2	0							
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0							
2	0							
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑						
2	0							
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑					
2	0							
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑				
2	0							
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖			
2	0							
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←		
2	0							
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	
2	0							
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0							
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0							
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑						
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖					
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←				
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑			
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑		
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0							
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑						
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑					
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑				
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖			
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←		
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0							
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖						
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑					
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑				
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑			
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑		
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0							
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑						
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑					
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑				
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖			
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑		
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	3 ←
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	3 ←
6	0							

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	3 ←
6	0	1 ↑						

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	3 ←
6	0	1 ↑	2 ↖					

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	3 ←
6	0	1 ↑	2 ↖	2 ←				

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	3 ←
6	0	1 ↑	2 ↖	2 ←	2 ↑			

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	3 ←
6	0	1 ↑	2 ↖	2 ←	2 ↑	2 ↑		

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	3 ←
6	0	1 ↑	2 ↖	2 ←	2 ↑	2 ↑	3 ↑	

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	3 ←
6	0	1 ↑	2 ↖	2 ←	2 ↑	2 ↑	3 ↑	4 ↖

Recording optimal solution

Store optimal choices in an additional array $b[i, j]$

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

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0 ↑	0 ↑	0 ↑	1 ↖	1 ←	1 ↖	1 ←
2	0	0 ↑	1 ↖	1 ←	1 ↑	1 ↑	1 ↑	2 ↖
3	0	0 ↑	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ↑
4	0	1 ↖	1 ↑	1 ↑	2 ↑	2 ↑	2 ↑	2 ↑
5	0	1 ↑	1 ↑	1 ↑	2 ↖	2 ↑	3 ↖	3 ←
6	0	1 ↑	2 ↖	2 ←	2 ↑	2 ↑	3 ↑	4 ↖

Longest common subsequence has length 4 and it is ABBA

Pseudocode and analysis

```
LCS-LENGTH(X, Y, m, n)
  let b[1..m, 1..n] and c[0..m, o..n] be new tables
  for i = 1 to m
      c[i, 0] = 0
  for j = 0 to n
      c[0, j] = 0
  for i = 1 to m
      for j = 1 to n
          if x_i == y_j
              c[i, j] = c[i − 1, j − 1] + 1
              b[i, j] = "↖"
          else if c[i − 1, j] ≥ c[i, j − 1]
                  c[i, j] = c[i − 1, j]
                  b[i, j] = "↑"
              else c[i, j] = c[i, j − 1]
                  b[i, j] = "←"
  return c and b
```

Pseudocode and analysis

```
LCS-LENGTH(X, Y, m, n)
  let b[1..m, 1..n] and c[0..m, o..n] be new tables
  for i = 1 to m
      c[i, 0] = 0
  for j = 0 to n
      c[0, j] = 0
  for i = 1 to m
      for j = 1 to n
          if x_i == y_j
              c[i, j] = c[i − 1, j − 1] + 1
              b[i, j] = "↖"
          else if c[i − 1, j] ≥ c[i, j − 1]
                  c[i, j] = c[i − 1, j]
                  b[i, j] = "↑"
              else c[i, j] = c[i, j − 1]
                  b[i, j] = "←"
  return c and b
```

- Time dominated by instructions inside the two nested loops which execute $m \cdot n$ times

Pseudocode and analysis

```
LCS-LENGTH(X, Y, m, n)
  let b[1..m, 1..n] and c[0..m, o..n] be new tables
  for i = 1 to m
      c[i, 0] = 0
  for j = 0 to n
      c[0, j] = 0
  for i = 1 to m
      for j = 1 to n
          if x_i == y_j
              c[i, j] = c[i − 1, j − 1] + 1
              b[i, j] = "↖"
          else if c[i − 1, j] ≥ c[i, j − 1]
                  c[i, j] = c[i − 1, j]
                  b[i, j] = "↑"
              else c[i, j] = c[i, j − 1]
                   b[i, j] = "←"
  return c and b
```

- Time dominated by instructions inside the two nested loops which execute $m \cdot n$ times
- Total time is $\Theta(m \cdot n)$.

Pseudocode and analysis for printing solution

```
PRINT-LCS(b, X, i, j)
  if i == 0 or j = 0
      return
  if b[i, j] == "↖"
      PRINT-LCS(b, X, i − 1, j − 1)
      print x_i
  elseif b[i, j] == "↑"
      PRINT-LCS(b, X, i − 1, j)
  else PRINT-LCS(b, X, i, j − 1)
```

Pseudocode and analysis for printing solution

```
PRINT-LCS(b, X, i, j)
  if i == 0 or j = 0
      return
  if b[i, j] == "↖"
      PRINT-LCS(b, X, i − 1, j − 1)
      print x_i
  elseif b[i, j] == "↑"
      PRINT-LCS(b, X, i − 1, j)
  else PRINT-LCS(b, X, i, j − 1)
```

- Each recursive call decreases $i + j$ by at least one.

Pseudocode and analysis for printing solution

```
PRINT-LCS(b, X, i, j)
  if i == 0 or j = 0
      return
  if b[i, j] == "↖"
      PRINT-LCS(b, X, i − 1, j − 1)
      print x_i
  elseif b[i, j] == "↑"
      PRINT-LCS(b, X, i − 1, j)
  else PRINT-LCS(b, X, i, j − 1)
```

- Each recursive call decreases $i + j$ by at least one.
- Hence, if we let $n = i + j$, the time needed is at most $T(n) \leq T(n-1) + \Theta(1)$ which is $O(n)$
- We can thus print the found string in time $\Theta(|X| + |Y|)$ (the lower bound following from that $T(n) \geq T(n-2) + \Theta(1)$)

Summary

- Identify choices and optimal substructure
- Write optimal solution recursively as a function of smaller subproblems
- Use top-down with memoization or bottom-up to solve the recursion efficiently (without repeatedly solving the same subproblems)

OPTIMAL BINARY SEARCH TREES

Lecture 11, 25.03.2025

Searching on Facebook

More popular than

Optimal binary search trees

- Given sequence $K = \langle k_1, k_2, \ldots, k_n \rangle$ of n distinct keys, sorted ($k_1 < k_2 < \cdots < k_n$).
- Want to build a binary search tree from the keys

Optimal binary search trees

- Given sequence $K = \langle k_1, k_2, \ldots, k_n \rangle$ of n distinct keys, sorted ($k_1 < k_2 < \cdots < k_n$).
- Want to build a binary search tree from the keys
- For k_i, have probability p_i that a search is for k_i
- Want BST with minimum expected search cost

Optimal binary search trees

- Given sequence $K = \langle k_1, k_2, \ldots, k_n \rangle$ of n distinct keys, sorted ($k_1 < k_2 < \cdots < k_n$).
- Want to build a binary search tree from the keys
- For k_i, have probability p_i that a search is for k_i
- Want BST with minimum expected search cost
- Actual cost = # of items examined

  For key k_i, cost = $\text{depth}_T(k_i) + 1$, where $\text{depth}_T(k_i)$ denotes the depth of k_i in BST T

Optimal binary search trees

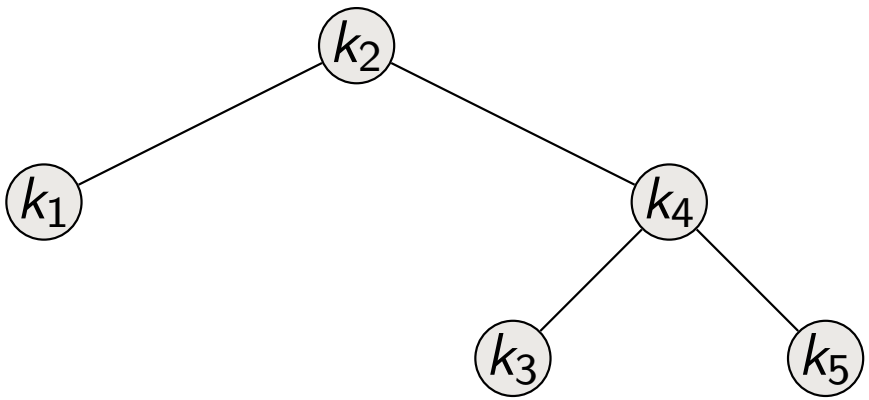
- Given sequence $K = \langle k_1, k_2, \ldots, k_n \rangle$ of n distinct keys, sorted ($k_1 < k_2 < \cdots < k_n$).
- Want to build a binary search tree from the keys
- For k_i, have probability p_i that a search is for k_i
- Want BST with minimum expected search cost
- Actual cost = # of items examined

  For key k_i, cost = $\text{depth}_T(k_i) + 1$, where $\text{depth}_T(k_i)$ denotes the depth of k_i in BST T

$$\begin{aligned}\mathbb{E}[\text{search cost in } T] &= \sum_{i=1}^{n} (\text{depth}_T(k_i) + 1) p_i \\ &= 1 + \sum_{i=1}^{n} \text{depth}_T(k_i) \cdot p_i\end{aligned}$$

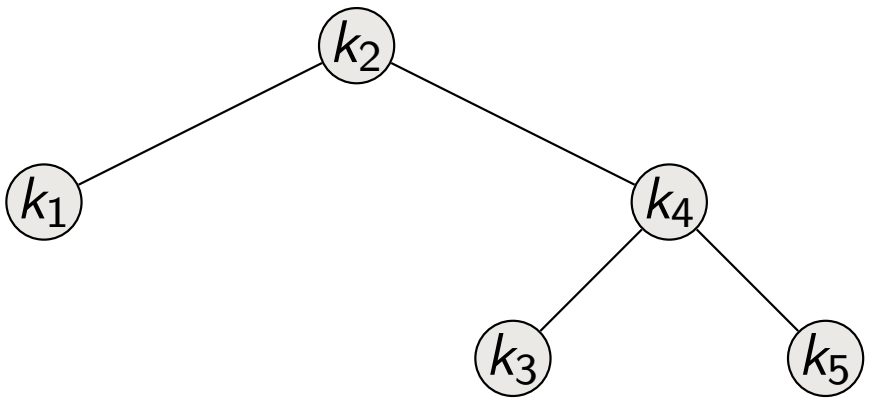
Example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3



Example

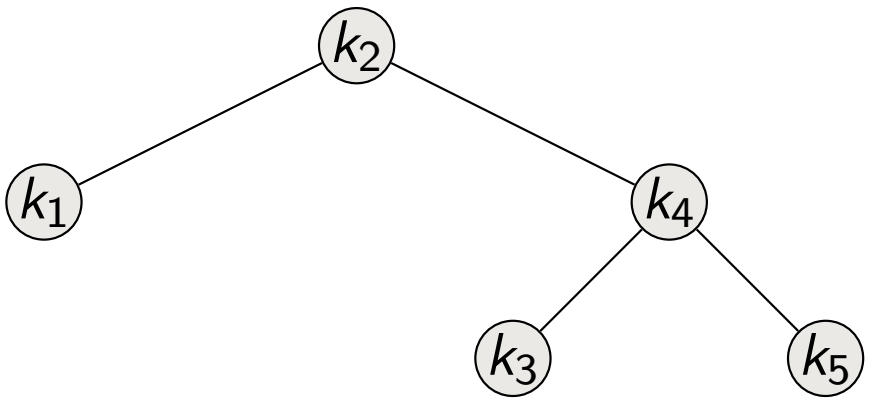
i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	2	.1
4	1	.2
5	2	.6
		1.15

Example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

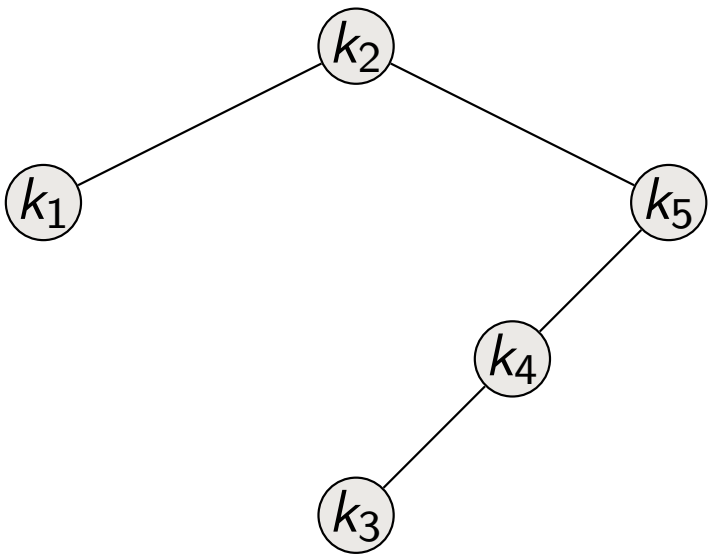


i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	2	.1
4	1	.2
5	2	.6
		1.15

Therefore, $\mathbb{E}[\text{search cost}] = 2.15$

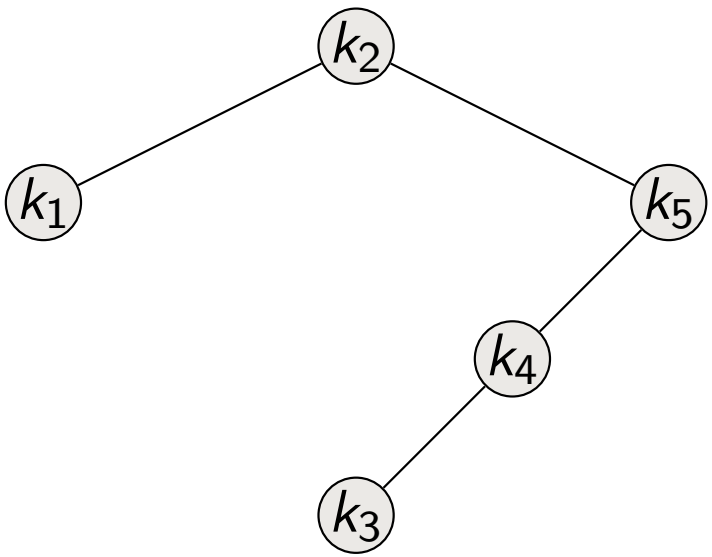
Example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3



Example

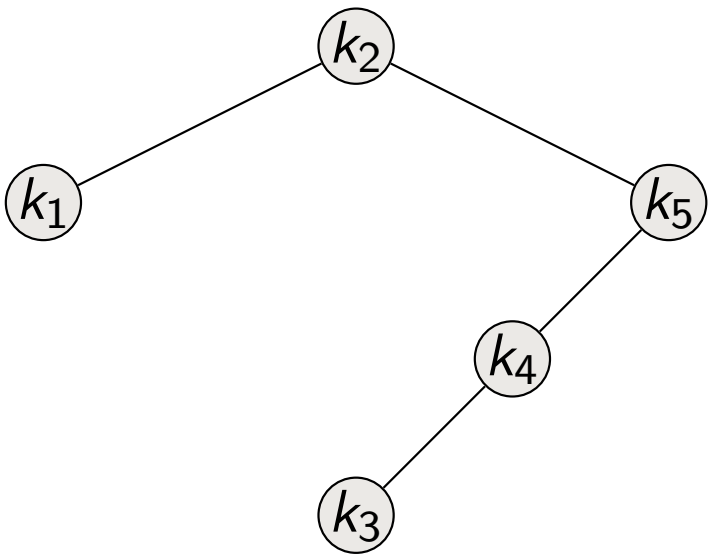
i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	3	.15
4	2	.4
5	1	.3
		1.10

Example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3



i	$\text{depth}_T(k_i)$	$\text{depth}_T(k_i) \cdot p_i$
1	1	.25
2	0	0
3	3	.15
4	2	.4
5	1	.3
		1.10

Therefore, $\mathbb{E}[\text{search cost}] = 2.10$, which turns out to be optimal

Observations

- Optimal BST might not have smallest height
- Optimal BST might not have highest-probability key at root

Observations

- Optimal BST might not have smallest height
- Optimal BST might not have highest-probability key at root

Build by exhaustive checking?

- Construct each n-node BST
- For each put in keys
- Then compute expected search cost

Observations

- Optimal BST might not have smallest height
- Optimal BST might not have highest-probability key at root

Build by exhaustive checking?

- Construct each n-node BST
- For each put in keys
- Then compute expected search cost
- But there are exponentially many trees

Observations

- Optimal BST might not have smallest height
- Optimal BST might not have highest-probability key at root

Build by exhaustive checking?

- Construct each n-node BST
- For each put in keys
- Then compute expected search cost
- But there are exponentially many trees

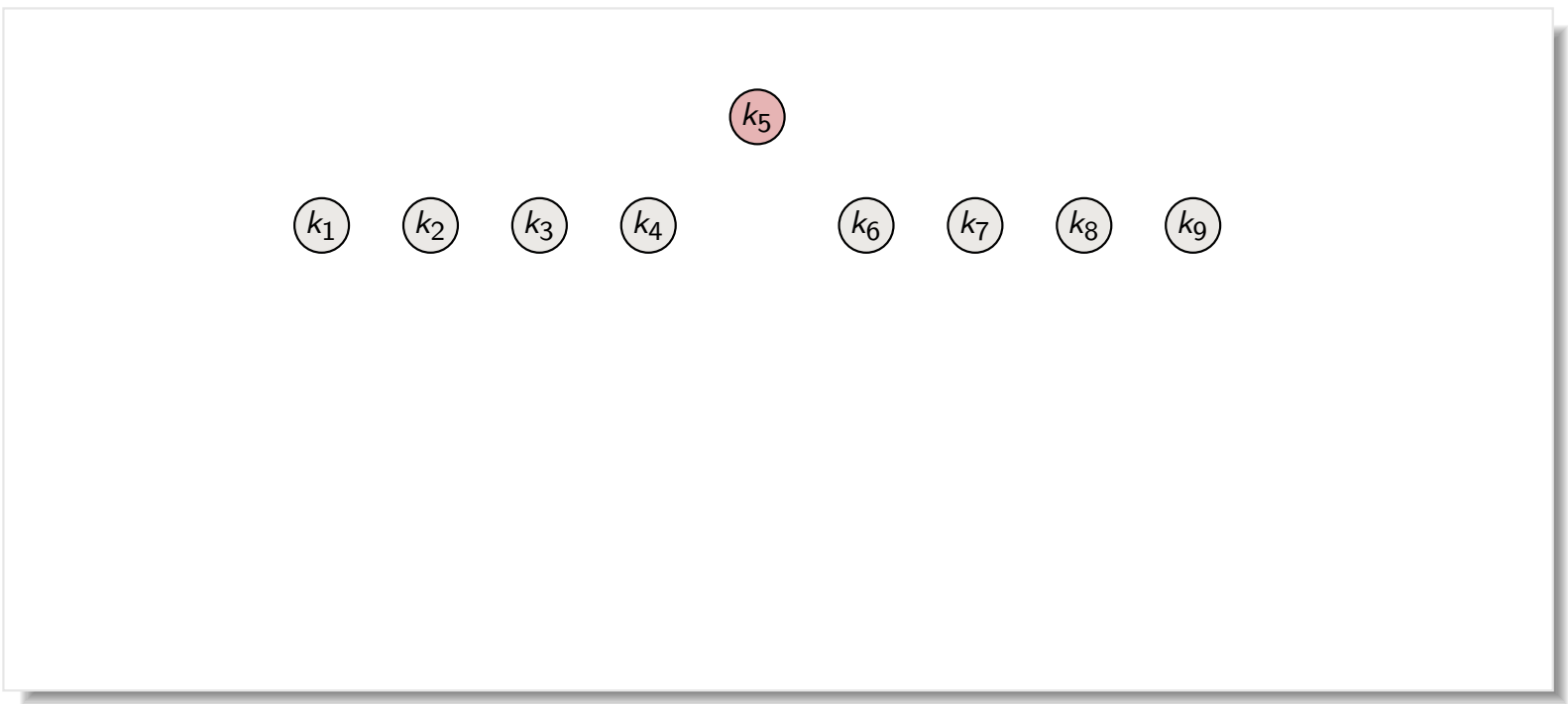
DP comes to the rescue :)

Optimal substructure

A binary search tree can be built by first picking the root and then building the subtrees recursively

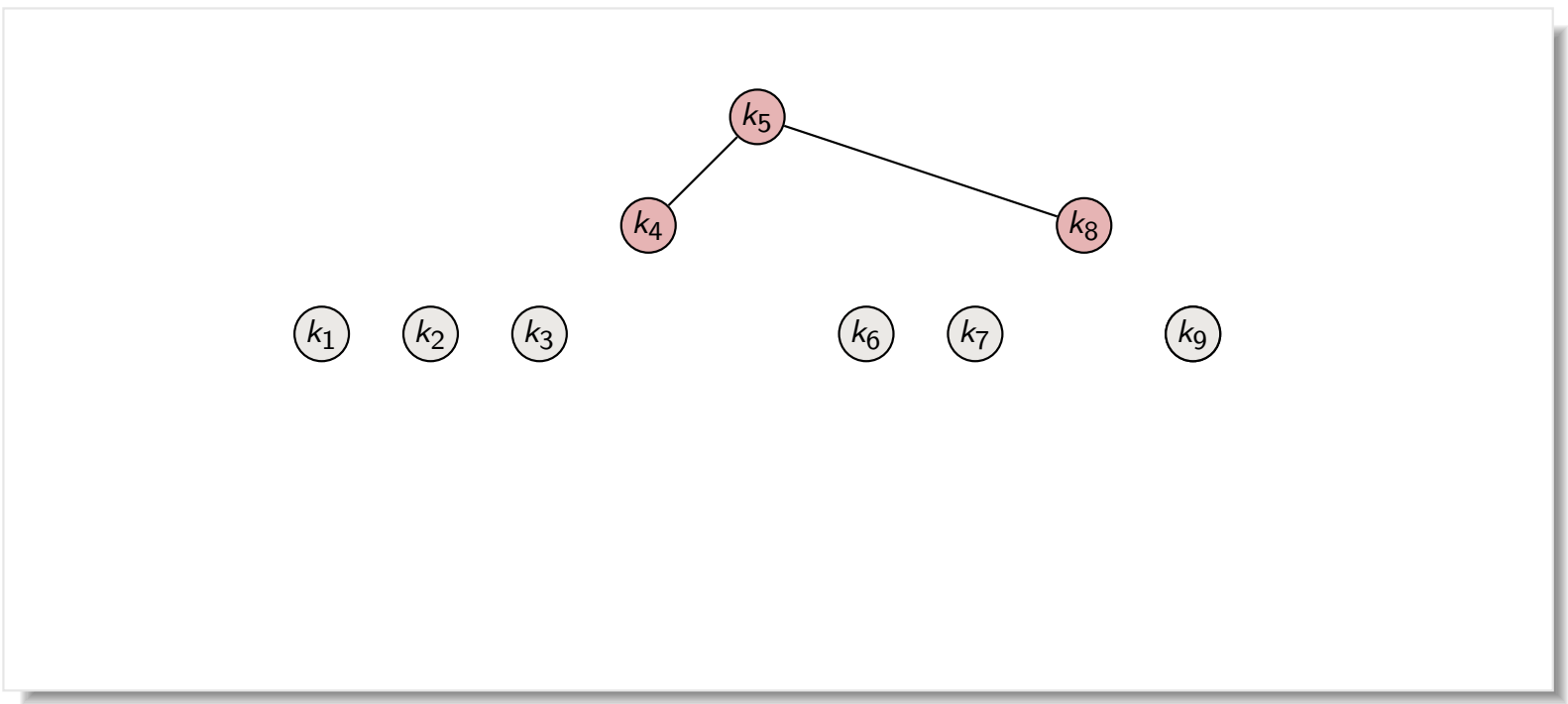
Optimal substructure

A binary search tree can be built by first picking the root and then building the subtrees recursively



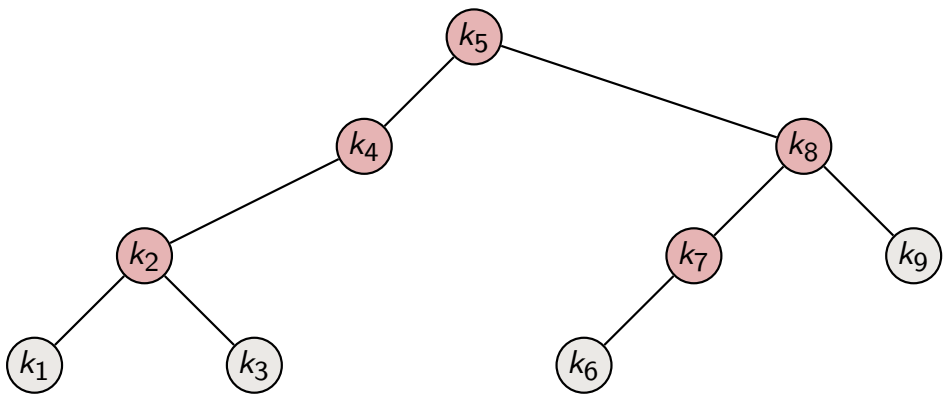
Optimal substructure

A binary search tree can be built by first picking the root and then building the subtrees recursively



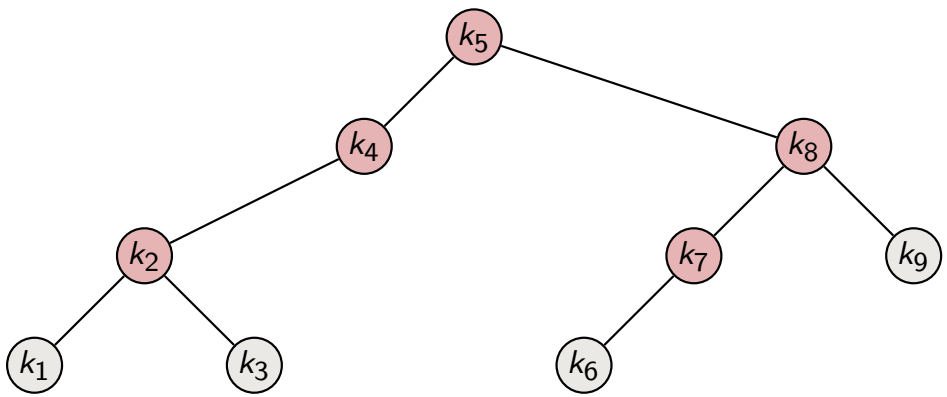
Optimal substructure

A binary search tree can be built by first picking the root and then building the subtrees recursively



Optimal substructure

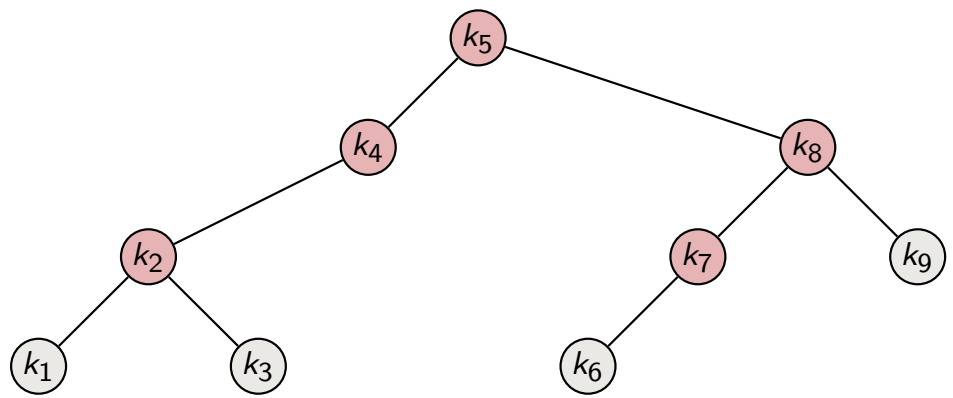
A binary search tree can be built by first picking the root and then building the subtrees recursively



$$\mathbb{E}[\text{search cost}] = p_5 + 2p_4 + 3p_2 + 4p_1 + 4p_3 + 2p_8 + 3p_7 + 3p_9 + 4p_6$$

Optimal substructure

A binary search tree can be built by first picking the root and then building the subtrees recursively



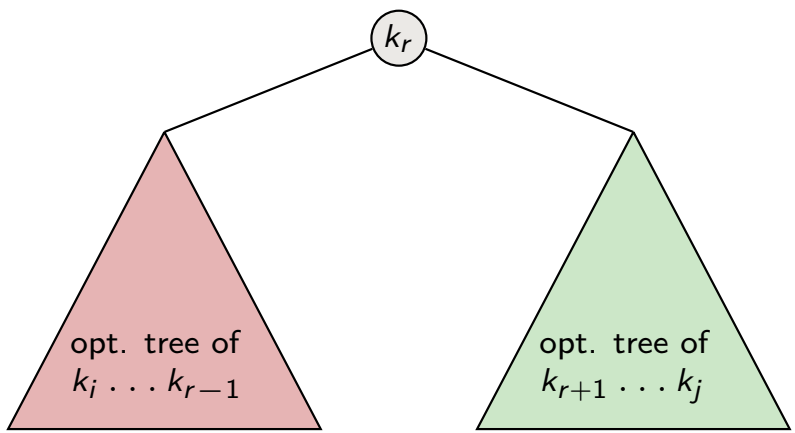
$$\begin{aligned}\mathbb{E}[\text{search cost}] = p_5 \\ &+ p_1 + p_2 + p_3 + p_4 + \mathbb{E}[\text{search cost left subtree}] \\ &+ p_6 + p_7 + p_8 + p_9 + \mathbb{E}[\text{search cost right subtree}]\end{aligned}$$

Optimal substructure

A binary search tree can be built by first picking the root and then building the subtrees recursively

After picking root solution to subtrees must be optimal

Build tree of nodes $k_i < k_{i+1} < \cdots < k_{j-1} < k_j$ by selecting best root r:



$\mathbb{E}[\text{search cost}] = p_r$

$+p_i + \cdots + p_{r-1} + \mathbb{E}[\text{search cost left subtree}]$

$+p_{r+1} + \cdots + p_j + \mathbb{E}[\text{search cost right subtree}]$

Recursive formulation

- Let $e[i, j] =$ expected search cost of optimal BST of $k_i \ldots k_j$

Recursive formulation

- Let $e[i, j] =$ expected search cost of optimal BST of $k_i \ldots k_j$

$$e[i, j] = \begin{cases}$$

Recursive formulation

- Let $e[i, j] =$ expected search cost of optimal BST of $k_i \ldots k_j$

$$e[i, j] = \begin{cases} & \text{if } i = j + 1 \end{cases}$$

Recursive formulation

- Let $e[i, j] =$ expected search cost of optimal BST of $k_i \ldots k_j$

$$e[i, j] = \begin{cases} 0 & \text{if } i = j + 1 \end{cases}$$

Recursive formulation

- Let $e[i, j] =$ expected search cost of optimal BST of $k_i \ldots k_j$

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ & \text{if } i \leq j \end{cases}$$

Recursive formulation

- Let $e[i, j]$ = expected search cost of optimal BST of $k_i \ldots k_j$

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i, r-1] + e[r+1, j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

Recursive formulation

- Let $e[i, j]$ = expected search cost of optimal BST of $k_i \ldots k_j$

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i, r-1] + e[r+1, j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

- Solve using bottom-up or top-down with memoization

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1						
2						
3						
4						
5						
6						

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0					
2		0				
3			0			
4				0		
5					0	
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25				
2		0				
3			0			
4				0		
5					0	
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25				
2		0	.2			
3			0			
4				0		
5					0	
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25				
2		0	.2			
3			0	.05		
4				0		
5					0	
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25				
2		0	.2			
3			0	.05		
4				0	.2	
5					0	
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25				
2		0	.2			
3			0	.05		
4				0	.2	
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65			
2		0	.2			
3			0	.05		
4				0	.2	
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65			
2		0	.2	.3		
3			0	.05		
4				0	.2	
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65			
2		0	.2	.3		
3			0	.05	.3	
4				0	.2	
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65			
2		0	.2	.3		
3			0	.05	.3	
4				0	.2	.7
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65	.8		
2		0	.2	.3		
3			0	.05	.3	
4				0	.2	.7
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65	.8		
2		0	.2	.3	.75	
3			0	.05	.3	
4				0	.2	.7
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65	.8		
2		0	.2	.3	.75	
3			0	.05	.3	.85
4				0	.2	.7
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65	.8	1.25	
2		0	.2	.3	.75	
3			0	.05	.3	.85
4				0	.2	.7
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65	.8	1.25	
2		0	.2	.3	.75	1.35
3			0	.05	.3	.85
4				0	.2	.7
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65	.8	1.25	2.1
2		0	.2	.3	.75	1.35
3			0	.05	.3	.85
4				0	.2	.7
5					0	.3
6						0

Bottom-up example

i	1	2	3	4	5
p_i	.25	.2	.05	.2	.3

$$e[i,j] = \begin{cases} 0 & \text{if } i = j+1 \\ \min_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + \sum_{\ell=i}^{j} p_\ell\} & \text{if } i \le j \end{cases}$$

e	0	1	2	3	4	5
1	0	.25	.65	.8	1.25	2.1
2		0	.2	.3	.75	1.35
3			0	.05	.3	.85
4				0	.2	.7
5					0	.3
6						0

Optimal BST has expected search cost 2.1
Can save decisions to reconstruct tree

Pseudocode of bottom-up

```
OPTIMAL-BST(p, q, n)
  let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
  for i = 1 to n + 1
      e[i, i − 1] = 0
      w[i, i − 1] = 0
  for l = 1 to n
      for i = 1 to n − l + 1
          j = i + l − 1
          e[i, j] = ∞
          w[i, j] = w[i, j − 1] + p_j
          for r = i to j
              t = e[i, r − 1] + e[r + 1, j] + w[i, j]
              if t < e[i, j]
                  e[i, j] = t
                  root[i, j] = r
  return e and root
```

$e[i,j]$ records the expected search cost of optimal BST of $k_i, \ldots, k_j$

$r[i,j]$ records the best root in optimal BST of $k_i, \ldots, k_j$

$w[i,j]$ records $\sum_{\ell=i}^{j} p_\ell$

Runtime Analysis

```
OPTIMAL-BST(p, q, n)
  let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
  for i = 1 to n + 1
      e[i, i − 1] = 0
      w[i, i − 1] = 0
  for l = 1 to n
      for i = 1 to n − l + 1
          j = i + l − 1
          e[i, j] = ∞
          w[i, j] = w[i, j − 1] + p_j
          for r = i to j
              t = e[i, r − 1] + e[r + 1, j] + w[i, j]
              if t < e[i, j]
                  e[i, j] = t
                  root[i, j] = r
  return e and root
```

Runtime Analysis

```
OPTIMAL-BST(p, q, n)
  let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
  for i = 1 to n + 1
      e[i, i − 1] = 0
      w[i, i − 1] = 0
  for l = 1 to n
      for i = 1 to n − l + 1
          j = i + l − 1
          e[i, j] = ∞
          w[i, j] = w[i, j − 1] + p_j
          for r = i to j
              t = e[i, r − 1] + e[r + 1, j] + w[i, j]
              if t < e[i, j]
                  e[i, j] = t
                  root[i, j] = r
  return e and root
```

- Runtime dominated by three nestled loops: total time is $\Theta(n^3)$

Runtime Analysis

```
OPTIMAL-BST(p, q, n)
  let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
  for i = 1 to n + 1
      e[i, i − 1] = 0
      w[i, i − 1] = 0
  for l = 1 to n
      for i = 1 to n − l + 1
          j = i + l − 1
          e[i, j] = ∞
          w[i, j] = w[i, j − 1] + p_j
          for r = i to j
              t = e[i, r − 1] + e[r + 1, j] + w[i, j]
              if t < e[i, j]
                  e[i, j] = t
                  root[i, j] = r
  return e and root
```

- Runtime dominated by three nestled loops: total time is $\Theta(n^3)$
- Alternatively, $\Theta(n^2)$ cells to fill in

Runtime Analysis

```
OPTIMAL-BST(p, q, n)
  let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
  for i = 1 to n + 1
      e[i, i − 1] = 0
      w[i, i − 1] = 0
  for l = 1 to n
      for i = 1 to n − l + 1
          j = i + l − 1
          e[i, j] = ∞
          w[i, j] = w[i, j − 1] + p_j
          for r = i to j
              t = e[i, r − 1] + e[r + 1, j] + w[i, j]
              if t < e[i, j]
                  e[i, j] = t
                  root[i, j] = r
  return e and root
```

- Runtime dominated by three nestled loops: total time is $\Theta(n^3)$
- Alternatively, $\Theta(n^2)$ cells to fill in
  Most cells take $\Theta(n)$ time to fill in

Runtime Analysis

```
OPTIMAL-BST(p, q, n)
  let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
  for i = 1 to n + 1
      e[i, i − 1] = 0
      w[i, i − 1] = 0
  for l = 1 to n
      for i = 1 to n − l + 1
          j = i + l − 1
          e[i, j] = ∞
          w[i, j] = w[i, j − 1] + p_j
          for r = i to j
              t = e[i, r − 1] + e[r + 1, j] + w[i, j]
              if t < e[i, j]
                  e[i, j] = t
                  root[i, j] = r
  return e and root
```

- Runtime dominated by three nestled loops: total time is $\Theta(n^3)$
- Alternatively, $\Theta(n^2)$ cells to fill in
  Most cells take $\Theta(n)$ time to fill in
  Hence, total time is $\Theta(n^3)$

Summary

- Identify choices and optimal substructure
- Write optimal solution recursively as a function of smaller subproblems
- Use top-down with memoization or bottom-up to solve the recursion efficiently (without repeatedly solving the same subproblems)
- Do a lot of exercises!