CSE 548: Analysis of Algorithms

Prerequisites Review 5
( Dynamic Programming )

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2019
The Rod Cutting Problem

Suppose you are given:

- a rod of length $n$ inches, and
- a list of prices $p_i$ for integer $i \in [1, n]$, where $p_i$ is the selling price of a rod of length $i$ inches.

Determine the maximum revenue $r_n$ obtainable by cutting up the rod and selling the pieces.
The Rod Cutting Problem

A sample price table for rods

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

Solve the problem for $n = 4$ and the price table given above.

<table>
<thead>
<tr>
<th>#pieces</th>
<th>#ways</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\left( \begin{array}{c} 3 \ 0 \end{array} \right) = 1$</td>
</tr>
<tr>
<td>2</td>
<td>$\left( \begin{array}{c} 3 \ 1 \end{array} \right) = 3$</td>
</tr>
<tr>
<td>3</td>
<td>$\left( \begin{array}{c} 3 \ 2 \end{array} \right) = 3$</td>
</tr>
<tr>
<td>4</td>
<td>$\left( \begin{array}{c} 3 \ 3 \end{array} \right) = 1$</td>
</tr>
</tbody>
</table>

Total: $8 = 2^3$

$\sum r_n = 5 + 5 = 10$
Rod Cutting: Standard Recursive Algorithm

A sample price table for rods

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

There is a different way of looking at the cuts and thus computing $r_n$.

$$r_n = \begin{cases} 0, & \text{if } n = 0, \\ \max_{1 \leq i \leq n} \{p_i + r_{n-i}\}, & \text{if } n > 0. \end{cases}$$
Rod Cutting: Standard Recursive Algorithm

\[ r_n = \begin{cases} 
0, & \text{if } n = 0, \\
\max_{1 \leq i \leq n} \{p_i + r_{n-i}\}, & \text{if } n > 0.
\end{cases} \]

**CUT-ROD** \((p, n)\)

1. \(\text{if } n = 0 \text{ then}\)
2. \(\text{return } 0\)
3. \(q \leftarrow -\infty\)
4. \(\text{for } i \leftarrow 1 \text{ to } n \text{ do}\)
5. \(q \leftarrow \max\{q, p[i] + \text{CUT-ROD}(p, n - i)\}\)
6. \(\text{return } q\)
**Rod Cutting: Standard Recursive Algorithm**

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.*
Rod Cutting: Standard Recursive Algorithm

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.*
Rod Cutting: Standard Recursive Algorithm

\[
\begin{align*}
&T(n) = \begin{cases}
\Theta(1), & \text{if } n = 0, \\
\sum_{i=1}^{n} T(n - i) + \Theta(1), & \text{if } n > 0.
\end{cases}
\end{align*}
\]

Solving: \( T(n) = \Theta(2^n). \)
When \texttt{CUT-ROD}( n ) is called with \( n = 5 \), the values of \( n \) passed to the recursive function calls are shown below.
Rod Cutting: Standard Recursive Algorithm

When \textbf{CUT-ROD}( n ) is called with \( n = 5 \), the values of \( n \) passed to the recursive function calls are shown below.

We are calling \textbf{CUT-ROD}( n ) or solving the problem for the same value of \( n \) over and over again!

How about saving the solution when we solve the problem for any given value of \( n \) for the first time?
Rod Cutting: Recursion with Memoization

**Memoized-Cut-Rod** (p, n)

1. \( r[0..n] \leftarrow \) new array
2. \( \text{for } i \leftarrow 0 \text{ to } n \text{ do} \)
3. \( r[i] \leftarrow -\infty \)
4. \( \text{return Memoized-Cut-Rod-Aux} (p, n, r) \)

**Memoized-Cut-Rod-Aux** (p, n, r)

1. \( \text{if } r[n] \geq 0 \text{ then} \)
2. \( \text{return } r[n] \)
3. \( \text{if } n = 0 \text{ then} \)
4. \( q \leftarrow 0 \)
3. \( \text{else } q \leftarrow -\infty \)
4. \( \text{for } i \leftarrow 1 \text{ to } n \text{ do} \)
5. \( q \leftarrow \max\{ q, p[i] + \text{Memoized-Cut-Rod-Aux} (p, n - i, r) \} \)
6. \( r[n] \leftarrow q \)
7. \( \text{return } q \)
Rod Cutting: Recursion with Memoization

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.*
Rod Cutting: Recursion with Memoization

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.*
Rod Cutting: Recursion with Memoization

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.*
Rod Cutting: Recursion with Memoization

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.*
**Rod Cutting: Recursion with Memoization**

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.*
Rod Cutting: Bottom-up Dynamic Programming

**BOTTOM-UP-CUT-ROD (p, n)**

1. \( r[0..n] \leftarrow \text{new array} \)
2. \( r[0] \leftarrow 0 \)
3. \( \text{for } j \leftarrow 1 \text{ to } n \text{ do} \)
4. \( q \leftarrow -\infty \)
5. \( \text{for } i \leftarrow 1 \text{ to } j \text{ do} \)
6. \( q \leftarrow \max\{ q, p[i] + r[j - i] \} \)
7. \( r[j] \leftarrow q \)
8. \( \text{return } r[n] \)
Rod Cutting: Bottom-up Dynamic Programming

**BOTTOM-UP-CUT-ROD ( p, n )**

1. \( r[0..n] \leftarrow \) new array
2. \( r[0] \leftarrow 0 \)
3. \( \text{for } j \leftarrow 1 \text{ to } n \text{ do} \)
4. \( q \leftarrow -\infty \)
5. \( \text{for } i \leftarrow 1 \text{ to } j \text{ do} \)
6. \( q \leftarrow \max\{ q, p[i] + r[j-i] \} \)
7. \( r[j] \leftarrow q \)
8. \( \text{return } r[n] \)
Rod Cutting: Bottom-up Dynamic Programming

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.*
Rod Cutting: Bottom-up Dynamic Programming

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.*

Stack overflow at $n = 248,108$
Rod Cutting: Recursive Divide-&-Conquer

**Divide-and-Conquer-Cut-Rod** (p, n)

1. r[0..n] ← new array
2. r[0] ← 0
3. for i ← 1 to n do
4. r[i] ← −∞
5. DC-Cut-Rod-A (p, r, 1, n)
6. return r[n]

**DC-Cut-Rod-Solve-Base** (p, r, k1, n1, k2, n2)

1. for j ← k2 to k2 + n2 − 1 do
2. q ← r[j]
3. for i ← k1 to min{j, k1 + n1 − 1} do
4. q ← max{q, p[i] + r[j − i]}
5. r[j] ← q
Rod Cutting: Recursive Divide-amp;Conquer

\[
\text{DC-CUT-Rod-A} (p, r, k, n)
\]

1. \text{if } n \leq \text{BASE\_SIZE} \text{ then}
2. \quad \text{DC-CUT-Rod-Solve-Base} (p, r, k, n, k, n)
3. \text{else}
4. \quad m \leftarrow \lfloor n/2 \rfloor
5. \quad \text{DC-CUT-Rod-A} (p, r, k, m)
6. \quad \text{DC-CUT-Rod-B} (p, r, k, m, k + m, n - m)
7. \quad \text{DC-CUT-Rod-A} (p, r, k + m, n - m)

// update left part using left part
// update right part using left part
// update right part using right part

update right part using data from left part
update left part using data from left part
split in the middle
update right part using data from right part
Rod Cutting: Recursive Divide-&-Conquer

\[ \text{DC-CUT-ROD-B} \ (p, r, k_1, n_1, k_2, n_2) \]

1. if \( n \leq \text{BASE\_SIZE} \) then
2. \( \text{DC-CUT-ROD-SOLVE-BASE} \ (p, r, k_1, n_1, k_2, n_2) \)
3. else
4. \( m_1 \leftarrow \lfloor n_1/2 \rfloor, m_2 \leftarrow \lfloor n_2/2 \rfloor \) // let \( L \equiv [k_1..k_1+n_1-1] \) and \( R \equiv [k_2..k_2+n_2-1] \)
5. \( \text{DC-CUT-ROD-B} \ (p, r, k_1, m_1, k_2, m_2) \) // left of \( L \) updates left of \( R \)
6. \( \text{DC-CUT-ROD-B} \ (p, r, k_1 + m_1, n_1 - m, k_2, m_2) \) // right of \( L \) updates left of \( R \)
7. \( \text{DC-CUT-ROD-B} \ (p, r, k_1, m_1, k_2 + m_2, n_2 - m_2) \) // left of \( L \) updates right of \( R \)
8. \( \text{DC-CUT-ROD-B} \ (p, r, k_1 + m_1, n_1 - m_1, k_2 + m_2, n_2 - m_2) \) // right of \( L \) updates right of \( R \)

\[ \begin{align*}
L & \equiv r[k_1..k_1+n_1-1] \\
R & \equiv r[k_2..k_2+n_2-1]
\end{align*} \]
Let $T(n)$, $T_A(n)$ and $T_B(n)$ be the running times of $DIVIDE-AND-CONQUER-CUT-ROD$, $DC-CUT-ROD-A$ and $DC-CUT-ROD-B$, respectively, on an input of size $n$. Then

$$T(n) = T_A(n) + \Theta(n).$$

$$T_A(n) = \begin{cases} \Theta(1), & \text{if } n \leq BASE\_SIZE, \\ 2T_A \left( \frac{n}{2} \right) + T_B \left( \frac{n}{2} \right) + \Theta(1), & \text{otherwise}. \end{cases}$$

$$T_B(n) = \begin{cases} \Theta(1), & \text{if } n \leq BASE\_SIZE, \\ 4T_B \left( \frac{n}{2} \right) + \Theta(1), & \text{otherwise}. \end{cases}$$

Solving: $T(n) = \Theta(n^2)$. 
Rod Cutting: Recursive Divide-&-Conquer

BASE_SIZE = 10,000

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.
BASE_SIZE = 10,000

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel E5-2650 with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM. Only one core was used.
Rod Cutting: Extracting the Solution

**EXTENDED-BOTTOM-UP-CUT-ROD (p, n)**

1. \( r[0..n] \leftarrow \) new array, \( s[0..n] \leftarrow \) new array
2. \( r[0] \leftarrow 0 \)
3. *for* \( j \leftarrow 1 \) *to* \( n \) *do*
4. \( q \leftarrow -\infty \)
5. *for* \( i \leftarrow 1 \) *to* \( j \) *do*
6. \( \text{if} \ q < p[i] + r[j - i] \text{ then} \)
7. \( q \leftarrow p[i] + r[j - i] \)
8. \( s[j] \leftarrow i \)
9. \( r[j] \leftarrow q \)
10. *return* \( r \) and \( s \)

**PRINT-CUT-ROD-SOLUTION (p, n)**

1. \((r, s) \leftarrow \text{EXTENDED-BOTTOM-UP-CUT-ROD (p, n)}\)
2. *while* \( n > 0 \) *do*
3. \( \text{print} \ s[n] \)
4. \( n \leftarrow n - s[n] \)
Rod Cutting: Extracting the Solution

A sample price table for rods

<table>
<thead>
<tr>
<th>length $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>price $p_i$</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>24</td>
<td>30</td>
</tr>
</tbody>
</table>

EXTENDED-BOTTOM-UP-CUT-ROD( $p$, $n$ ) returns the following arrays:

<table>
<thead>
<tr>
<th>$i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r[i]$</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>13</td>
<td>17</td>
<td>18</td>
<td>22</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>$s[i]$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>
A $p \times q$ matrix $A$ and a $q' \times r$ matrix $B$ can be multiplied provided $q = q'$.

The result will be a $p \times r$ matrix $C$. 
Matrix-Chain Multiplication

\[
\begin{array}{ccc}
A & \times & B \\
(p \times q) & & (q' \times r) \\
\end{array}
\]

\[
C \\
(p \times r)
\]

provided \( q = q' \)

**Matrix-Multiply** \((p, q, A, q', r, B)\)

1. if \( q \neq q' \) then
2. error “incompatible dimensions”
3. else
4. \( C \leftarrow \) new \( p \times r \) matrix
5. for \( i \leftarrow 1 \) to \( p \) do
6. for \( j \leftarrow 1 \) to \( r \) do
7. \( C[i,j] \leftarrow 0 \)
8. for \( k \leftarrow 1 \) to \( q \) do
9. \( C[i,j] \leftarrow C[i,j] + A[i,k] \times B[k,j] \)
10. return \( C \)

Time needed to multiply the \( p \times q \) matrix \( A \) and the \( q \times r \) matrix \( B \) is dominated by the total number \( pqr \) of scalar multiplications performed in line 7.

Hence, running time of the algorithm is \( \Theta(pqr) \).
We can multiply the four matrices on the left hand side in five distinct orders.
Matrix-Chain Multiplication

\[
\begin{align*}
A_1 & \quad (3 \times 4) \\
A_2 & \quad (4 \times 5) \\
A_3 & \quad (5 \times 2) \\
A_4 & \quad (2 \times 6)
\end{align*}
\]

number of scalar multiplications

\[= 5 \times 2 \times 6 + 4 \times 5 \times 6 + 3 \times 4 \times 6\]

\[= 252\]
Matrix-Chain Multiplication

\[ A_1 \]  
\[(3 \times 4)\]

\[ A_2 \]  
\[(4 \times 5)\]

\[ A_3 \]  
\[(5 \times 2)\]

\[ A_4 \]  
\[(2 \times 6)\]

number of scalar multiplications

\[ = 4 \times 5 \times 2 + 4 \times 2 \times 6 + 3 \times 4 \times 6 \]
\[ = 160 \]
Matrix-Chain Multiplication

\[
A_1 \begin{pmatrix} 3 & \times & 4 \end{pmatrix} \times \begin{pmatrix} 4 & \times & 5 \end{pmatrix} \times \begin{pmatrix} 5 & \times & 2 \end{pmatrix} \times \begin{pmatrix} 2 & \times & 6 \end{pmatrix}
\]

number of scalar multiplications
\[
= 3 \times 4 \times 5 + 5 \times 2 \times 6 + 3 \times 5 \times 6
= 210
\]
Matrix-Chain Multiplication

$A_1 (3 \times 4) \times A_2 (4 \times 5) \times A_3 (5 \times 2) \times A_4 (2 \times 6)$

number of scalar multiplications

$$= 4 \times 5 \times 2 + 3 \times 4 \times 2 + 3 \times 2 \times 6$$

$$= 100$$
Matrix-Chain Multiplication

\[
\begin{align*}
A_1 &= (3 \times 4) \\
A_2 &= (4 \times 5) \\
A_3 &= (5 \times 2) \\
A_4 &= (2 \times 6)
\end{align*}
\]

number of scalar multiplications
\[
= 3 \times 4 \times 5 + 3 \times 5 \times 2 + 3 \times 2 \times 6
\]
\[
= 126
\]
Matrix-Chain Multiplication

The matrix-chain multiplication problem:

Given 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 $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 \ldots A_n$ in a way that minimizes the number of scalar multiplications.
Matrix-Chain Multiplication

Let $P(n) = \text{number of parenthesizations of a sequence of } n \text{ matrices.}$ Then

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

Very easy to show that $P(n) = \Omega(2^n)$.

Hence, exhaustively checking all possible parenthesizations of the given chain of matrices does not give an efficient algorithm.
Matrix-Chain Mult: Standard Recursive Algorithm

Let $A_{i...j} = A_i A_{i+1} ... A_{j-1} A_j$ for $1 \leq i \leq j \leq n$.

Let $m(i, j) =$ the minimum number of scalar multiplications needed to compute the matrix $A_{i...j}$.

Then $m(1, n) =$ the minimum number of scalar multiplications needed to compute $A_{1...n}$ (i.e., solve the entire problem).

$$m(i, j) = \begin{cases} 0, & \text{if } i = j, \\ \min_{i \leq k < j} \{ m(i, k) + m(k + 1, j) + p_{i-1} p_k p_j \}, & \text{if } i < j. \end{cases}$$
**Matrix-Chain Mult: Standard Recursive Algorithm**

\[
\text{RECURSIVE-MATRIX-CHAIN} \left( p, i, j \right)
\]

1. \( \text{if } i = j \text{ then} \)
2. \( \text{return } 0 \)
3. \( q \leftarrow \infty \)
4. \( \text{for } k \leftarrow i \text{ to } j - 1 \text{ do} \)
5. \( q \leftarrow \min \left( q, \begin{array}{c}
\text{RECURSIVE-MATRIX-CHAIN} \left( p, i, k \right) \\
+ \text{RECURSIVE-MATRIX-CHAIN} \left( p, k + 1, j \right) \\
+ p_{i-1}p_kp_j
\end{array} \right) \)
6. \( \text{return } q \)
Matrix-Chain Mult: Standard Recursive Algorithm

Let $T(n)$ be the running time of the algorithm on an input of size $n$. Then

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

Solving: $T(n) \geq 2^{n-1} \Rightarrow T(n) = \Omega(2^n)$. 
Matrix-Chain Mult: Standard Recursive Algorithm

```
  1..4
 /   \
/     \  
1..1   2..4
       /  \
      /    \
    2..2  3..4  2..3  4..4  1..1  2..2  3..3  4..4
       /  \
      /    \
    3..3  4..4  2..2  3..3  2..2  3..3  1..1  2..2
```
Matrix-Chain Mult: Standard Recursive Algorithm
Matrix-Chain Mult: Recursion with Memoization

**Memoized-Matrix-Chain (p)**

1. \( n \leftarrow p.length - 1 \)
2. \( m[1..n, 1..n] \leftarrow \) new table
3. \( \text{for } i \leftarrow 1 \text{ to } n \text{ do} \)
4. \( \text{for } j \leftarrow i \text{ to } n \text{ do} \)
5. \( m[i, j] \leftarrow \infty \)
6. \( \text{return Lookup-Chain (m, p, 1, n)} \)

**Lookup-Chain (m, p, i, j)**

1. \( \text{if } m[i, j] < \infty \text{ then} \)
2. \( \text{return } m[i, j] \)
3. \( \text{if } i = j \text{ then} \)
4. \( m[i, j] \leftarrow 0 \)
5. \( \text{for } k \leftarrow i \text{ to } j - 1 \text{ do} \)
6. \( q \leftarrow \text{Lookup-Chain (m, p, i, k)} \)
   + \( \text{Lookup-Chain (m, p, k + 1, j)} \)
   + \( p_{i-1}p_kp_j \)
7. \( \text{if } q < m[i, j] \text{ then} \)
8. \( m[i, j] \leftarrow q \)
9. \( \text{return } m[i, j] \)
Matrix-Chain Mult: Bottom-up DP

Matrix-Chain-Order (p)

1. \( n \leftarrow p.length - 1 \)
2. \( m[1..n, 1..n] \leftarrow \text{new table}, s[1..n - 1, 2..n] \leftarrow \text{new table} \)
3. \( \text{for } i \leftarrow 1 \text{ to } n \text{ do} \)
4. \( m[i, i] \leftarrow 0 \)
5. \( \text{for } l \leftarrow 2 \text{ to } n \text{ do} \quad // l \text{ is the chain length} \)
6. \( \quad \text{for } i \leftarrow 1 \text{ to } n - l + 1 \text{ do} \)
7. \( j \leftarrow i + l - 1 \)
8. \( m[i, j] \leftarrow \infty \)
9. \( \quad \text{for } k \leftarrow i \text{ to } j - 1 \text{ do} \)
10. \( q \leftarrow m[i, k] + m[k + 1, j] + p_{i-1}p_ip_j \)
11. \( \quad \text{if } q < m[i, j] \text{ then} \)
12. \( \quad m[i, j] \leftarrow q \)
13. \( \quad s[i, j] \leftarrow k \)
14. \( \text{return } m \text{ and } s \)
# Matrix-Chain Mult: Bottom-up DP

<table>
<thead>
<tr>
<th>matrix</th>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$A_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimension</td>
<td>$30 \times 35$</td>
<td>$35 \times 15$</td>
<td>$15 \times 5$</td>
<td>$5 \times 10$</td>
<td>$10 \times 20$</td>
<td>$20 \times 25$</td>
</tr>
</tbody>
</table>

![Matrix-Chain Mult: Bottom-up DP Diagram](image)
Matrix-Chain Mult: Extracting the Solution

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

1. if $i = j$ then
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 “)”
Matrix-Chain Mult: Recursive Divide-&-Conquer
Matrix-Chain Mult: Recursive Divide-&-Conquer
Matrix-Chain Mult: Recursive Divide-&-Conquer

\[ A_{par}(\langle X, X, X \rangle) \]
1. if \( X \) is a small matrix then \( A_{loop-par}(\langle X, X, X \rangle) \)
2. else
3. \( \textbf{par: } A_{par}(\langle X_{11}, X_{11}, X_{11} \rangle), A_{par}(\langle X_{22}, X_{22}, X_{22} \rangle) \)
4. \( B_{par}(\langle X_{12}, X_{11}, X_{22} \rangle) \)

\[ B_{par}(\langle X, U, V \rangle) \]
1. if \( X \) is a small matrix then \( B_{loop-par}(\langle X, U, V \rangle) \)
2. else
3. \( B_{par}(\langle X_{21}, U_{22}, V_{11} \rangle) \)
4. \( \textbf{par: } C_{par}(\langle X_{11}, U_{12}, V_{21} \rangle), C_{par}(\langle X_{22}, X_{21}, V_{12} \rangle) \)
5. \( \textbf{par: } B_{par}(\langle X_{11}, U_{11}, V_{11} \rangle), B_{par}(\langle X_{22}, X_{22}, V_{22} \rangle) \)
6. \( C_{par}(\langle X_{12}, U_{12}, X_{22} \rangle) \)
7. \( C_{par}(\langle X_{12}, X_{11}, V_{12} \rangle) \)
8. \( B_{par}(\langle X_{12}, U_{11}, V_{22} \rangle) \)

\[ C_{par}(\langle X, U, V \rangle) \]
1. if \( X \) is a small matrix then \( C_{loop-par}(\langle X, U, V \rangle) \)
2. else
3. \( \textbf{par: } C_{par}(\langle X_{11}, U_{11}, V_{11} \rangle), C_{par}(\langle X_{12}, U_{11}, V_{12} \rangle), C_{par}(\langle X_{21}, U_{21}, V_{11} \rangle), C_{par}(\langle X_{22}, U_{21}, V_{12} \rangle) \)
4. \( \textbf{par: } C_{par}(\langle X_{11}, U_{12}, V_{21} \rangle), C_{par}(\langle X_{12}, U_{12}, V_{22} \rangle), C_{par}(\langle X_{21}, U_{22}, V_{21} \rangle), C_{par}(\langle X_{22}, U_{22}, V_{22} \rangle) \)
Matrix-Chain Mult: Empirical Performance

R-DP: recursive divide-&-conquer (BASE_SIZE = 64 × 64),
I-DP: iterative DP, Tiled I-DP: tiled iterative DP

*Run on a dual-socket (2 × 8 cores) 2.7 GHz Intel Sandy Bridge with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM.
Matrix-Chain Mult: Empirical Performance

R-DP: recursive divide-\&-conquer (BASE\_SIZE = 64 × 64),
I-DP: iterative DP, Tiled I-DP: tiled iterative DP

*Run on a dual-socket (2 × 8 cores) 2.7 GHz Intel Sandy Bridge with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM.*
Matrix-Chain Mult: Empirical Performance

R-DP: recursive divide-&-conquer (BASE_SIZE = 64 × 64),
I-DP: iterative DP, Tiled I-DP: tiled iterative DP

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel Sandy Bridge with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM.
Matrix-Chain Mult: Empirical Performance

R-DP: recursive divide-&-conquer (BASE_SIZE = 64 × 64),
I-DP: iterative DP, Tiled I-DP: tiled iterative DP

*Run on a dual-socket (2 × 8 cores) 2.7 GHz Intel Sandy Bridge with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM.
**Matrix-Chain Multi: Empirical Performance**

- **R-DP:** recursive divide-&-conquer (BASE_SIZE = $64 \times 64$),
- **I-DP:** iterative DP,
- **Tiled I-DP:** tiled iterative DP

---

*Run on a dual-socket (2 × 8 cores) 2.7 GHz Intel Sandy Bridge with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM.*
Matrix-Chain Mult: Empirical Performance

R-DP: recursive divide-&-conquer (BASE_SIZE = 64 × 64),  
I-DP: iterative DP,  Tiled I-DP: tiled iterative DP

*Run on a dual-socket (2 × 8 cores) 2.7 GHz Intel Sandy Bridge with private 32KB L1 and 256KB L2 caches,  a shared 20MB L3 cache per socket and 32GB RAM.*
Matrix-Chain Mult: Empirical Performance

R-DP: recursive divide-&-conquer (BASE_SIZE = 64 × 64),
I-DP: iterative DP, Tiled I-DP: tiled iterative DP

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel Sandy Bridge with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM.
Matrix-Chain Mult: Empirical Performance

**R-DP:** recursive divide-&-conquer (BASE_SIZE = $64 \times 64$),
**I-DP:** iterative DP, **Tiled I-DP:** tiled iterative DP

---

*Run on a dual-socket (2 × 8 cores) 2.0 GHz Intel Sandy Bridge with private 32KB L1 and 256KB L2 caches, a shared 20MB L3 cache per socket and 32GB RAM.*
Dynamic Programming vs. Divide-and-Conquer

- Dynamic programming, like the divide-and-conquer method, solves problems by combining solutions to subproblems.

- Divide-and-conquer algorithms:
  - partition the problem into disjoint subproblems,
  - solve the subproblems recursively, and
  - then combine their solutions to solve the original problem.

- In contrast, dynamic programming applies when the subproblems overlap — that is, when subproblems share subsubproblems.

- A dynamic-programming algorithm solves each subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each subsubproblem.
Elements of Dynamic Programming

An optimization problem must have the following two ingredients for dynamic programming to apply.

1) Optimal substructure
   - an optimal solution to the problem contains within it optimal solutions to subproblems

2) Overlapping subproblems
   - subproblems share subsubproblems and/or subsubsubproblems and/or subsubsubsubproblems, and so on
Dynamic Programming

When developing a dynamic-programming algorithm, we follow a sequence of four steps:

1) Characterize the structure of an optimal solution.
2) Recursively define the value of an optimal solution.
3) Compute the value of an optimal solution, typically in a bottom-up fashion.
4) Construct an optimal solution from computed information.

If we need only the value of an optimal solution, and not the solution itself, then we can omit step 4.

If we perform step 4, we sometimes maintain additional information during step 3 so that we can easily construct an optimal solution.
Longest Common Subsequence (LCS)

A *subsequence* of a sequence $X$ is obtained by deleting zero or more symbols from $X$.

Example:

$X = abcba$

$Z = bca \iff$ obtained by deleting the 1\textsuperscript{st} ‘$a$’ and the 2\textsuperscript{nd} ‘$b$’ from $X$

A *Longest Common Subsequence (LCS)* of two sequence $X$ and $Y$ is a sequence $Z$ that is a subsequence of both $X$ and $Y$, and is the longest among all such subsequences.

Given $X$ and $Y$, the *LCS problem* asks for such a $Z$. 
LCS: Optimal Substructure

Given two sequences: \( X = \langle x_1, x_2, \ldots, x_m \rangle \) and \( Y = \langle y_1, y_2, \ldots, y_n \rangle \)

Let \( Z = \langle z_1, z_2, \ldots, z_k \rangle \) be any LCS of \( X \) and \( Y \).

For \( 0 \leq i \leq m \), let \( X_i = \langle x_1, x_2, \ldots, x_i \rangle \). We define \( Y_i \) and \( Z_i \) similarly.

Then

1. If \( x_m = y_n \),
   
   then \( z_k = x_m = y_n \) and \( Z_{k-1} \) is an LCS of \( X_{m-1} \) and \( Y_{n-1} \).

2. If \( x_m \neq y_n \),
   
   then \( z_k \neq x_m \) implies that \( Z \) is an LCS of \( X_{m-1} \) and \( Y \).

3. If \( x_m \neq y_n \),
   
   then \( z_k \neq y_n \) implies that \( Z \) is an LCS of \( X \) and \( Y_{n-1} \).
**LCS: Recurrence**

Given two sequences: \( X = \langle x_1, x_2, \ldots, x_m \rangle \) and \( Y = \langle y_1, y_2, \ldots, y_n \rangle \)

For \( 0 \leq i \leq m \) and \( 0 \leq j \leq n \),

let \( c[i,j] \) be the length of an LCS of \( X_i \) and \( Y_j \). Then

\[
c[i,j] = \begin{cases} 
0, & \text{if } i = 0 \lor j = 0, \\
& \text{otherwise.} \\
c[i-1,j-1] + 1, & \text{if } i,j > 0 \land x_i = y_j, \\
\max\{c[i,j-1], c[i-1,j]\}, & \text{otherwise.}
\end{cases}
\]
**LCS: Bottom-up DP**

**LCS-LENGTH** (X, Y)

1. \( m \leftarrow X.\) length
2. \( n \leftarrow Y.\) length
3. \( b[1 \ldots m, 1 \ldots n] \leftarrow \) new table, \( c[0 \ldots m, 0 \ldots n] \leftarrow \) new table
4. \( \text{for } i \leftarrow 1 \) to \( m \)
5. \( c[i, 0] \leftarrow 0 \)
6. \( \text{for } j \leftarrow 0 \) to \( n \)
7. \( c[0, j] \leftarrow 0 \)
8. \( \text{for } i \leftarrow 1 \) to \( m \)
9. \( \text{for } j \leftarrow 1 \) to \( n \)
10. \( \text{if } x_i = y_j \)
11. \( c[i, j] \leftarrow c[i - 1, j - 1] + 1 \)
12. \( b[i, j] \leftarrow \text{“\textbackslash\textbackslash”} \)
13. \( \text{elseif } c[i - 1, j] \geq c[i, j - 1] \)
14. \( c[i, j] \leftarrow c[i - 1, j] \)
15. \( b[i, j] \leftarrow \text{“\uparrow”} \)
16. \( \text{else } c[i, j] \leftarrow c[i, j - 1] \)
17. \( b[i, j] \leftarrow \text{“\leftarrow”} \)

**Running time = \( \Theta(mn) \)**
LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td>y_j</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>$y_j$</th>
<th>$B$</th>
<th>$D$</th>
<th>$C$</th>
<th>$A$</th>
<th>$B$</th>
<th>$A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td><strong>y_j</strong></td>
<td><strong>B</strong></td>
<td><strong>D</strong></td>
<td><strong>C</strong></td>
<td><strong>A</strong></td>
<td><strong>B</strong></td>
<td><strong>A</strong></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>← 1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LCS:** Longest Common Subsequence
## LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>y_j</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>← 1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>← 1</td>
<td>← 1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## LCS: Bottom-up DP

<table>
<thead>
<tr>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>y_j</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>↖</td>
<td>1</td>
<td>←</td>
<td>1</td>
<td>←</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>↑</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>←</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**LCS: Bottom-up DP**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↘</td>
<td>1</td>
<td>← 1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>↘</td>
<td>1</td>
<td>← 1</td>
<td>← 1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>↑</td>
<td>↑</td>
<td>↘</td>
<td>1</td>
<td>2</td>
<td>← 2</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>↘</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td><strong>y_j</strong></td>
<td><strong>B</strong></td>
<td><strong>D</strong></td>
<td><strong>C</strong></td>
<td><strong>A</strong></td>
<td><strong>B</strong></td>
<td><strong>A</strong></td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>↖</td>
<td>1</td>
<td>←</td>
<td>1</td>
<td>←</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td>↖</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td>↑</td>
<td>↖</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### LCS: Bottom-up DP

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>B</th>
<th>D</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>(\searrow)</td>
<td>(\leftarrow)</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>(\searrow)</td>
<td>1</td>
<td>(\leftarrow)</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>↑</td>
<td>↑</td>
<td>(\searrow)</td>
<td>2</td>
<td>(\leftarrow)</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>(\searrow)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>↑</td>
<td>(\searrow)</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>(\searrow)</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**LCS Matrix**:
- **Bottom-up DP** approach to find the Longest Common Subsequence (LCS) between two sequences.
## LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↘</td>
<td>← 1</td>
<td>← 1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>↘</td>
<td>1</td>
<td>← 1</td>
<td>← 1</td>
<td>1</td>
<td>← 2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↘</td>
<td>2</td>
<td>← 2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>↘</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>↑</td>
<td>↘</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↘</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>↘</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
### LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td><strong>x_i</strong></td>
<td><strong>y_j</strong></td>
<td><strong>B</strong></td>
<td><strong>D</strong></td>
<td><strong>C</strong></td>
<td><strong>A</strong></td>
<td><strong>B</strong></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↘</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>↗</td>
<td>1</td>
<td>← 1</td>
<td>← 1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↘</td>
<td>2</td>
<td>← 2</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td>↗</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td>↑</td>
<td>↗</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↗</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td>↗</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
### LCS: Bottom-up DP

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
<th>$y_j$</th>
<th>$B$</th>
<th>$D$</th>
<th>$C$</th>
<th>$A$</th>
<th>$B$</th>
<th>$\text{A}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$x_i$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>$A$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$B$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$C$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$B$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$D$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$\text{A}$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>$B$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

The shaded cell indicates the length of the longest common subsequence (LCS) at that point.
## LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>y_j</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>y_j</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>← 1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>← 1</td>
<td>← 1</td>
<td>1</td>
<td>2</td>
<td>← 2</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>← 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>← 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>← 3</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LCS Matrix**

**Path:** B → A → B
## LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td><strong>x_i</strong></td>
<td><strong>y_j</strong></td>
<td><strong>B</strong></td>
<td><strong>D</strong></td>
<td><strong>C</strong></td>
<td><strong>A</strong></td>
<td><strong>B</strong></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>← 1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>← 1</td>
<td>← 1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>← 2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
# LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td></td>
<td><strong>y_j</strong></td>
<td><strong>B</strong></td>
<td><strong>D</strong></td>
<td><strong>C</strong></td>
<td><strong>A</strong></td>
<td><strong>B</strong></td>
</tr>
<tr>
<td><strong>x_i</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>1</td>
<td>← 1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>↖</td>
<td>1</td>
<td>← 1</td>
<td>← 1</td>
<td>2</td>
<td>← 2</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>2</td>
<td>← 2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>↖</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>↑</td>
<td>↖</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>↖</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
## LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y_j</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>j</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Bottom-up DP**

  - **LCS:** Longest Common Subsequence
  - **DP:** Dynamic Programming

  - **Matrix:**
    - **Rows:** x_i (input sequence)
    - **Columns:** y_j (output sequence)

  - **Cell Values:**
    - **0:** Initial value
    - **1:** Match
    - **2:** Mismatch
    - **3:** Insertion
    - **4:** Deletion

  - **Diagonal Moves:**
    - **Up:** Match
    - **Left:** Insertion
    - **Right:** Deletion

  - **Diagonal Moves:**
    - **Up:** Match
    - **Left:** Insertion
    - **Right:** Deletion

  - **Diagonal Moves:**
    - **Up:** Match
    - **Left:** Insertion
    - **Right:** Deletion

  - **Diagonal Moves:**
    - **Up:** Match
    - **Left:** Insertion
    - **Right:** Deletion

  - **Diagonal Moves:**
    - **Up:** Match
    - **Left:** Insertion
    - **Right:** Deletion

  - **Diagonal Moves:**
    - **Up:** Match
    - **Left:** Insertion
    - **Right:** Deletion

  - **Diagonal Moves:**
    - **Up:** Match
    - **Left:** Insertion
    - **Right:** Deletion

  - **Diagonal Moves:**
    - **Up:** Match
    - **Left:** Insertion
    - **Right:** Deletion
## LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>j</strong></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td><strong>y_j</strong></td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td><strong>x_i</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>1</td>
<td>←</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>←</td>
<td>←</td>
<td>↑</td>
<td>1</td>
<td>2</td>
<td>←</td>
</tr>
<tr>
<td>3</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>2</td>
<td>←</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>↑</td>
<td>↖</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>↖</td>
<td>↑</td>
<td>↑</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**LCS Bottom-up DP**

**Algorithm:**
- Initialize the table with zeros.
- For each cell (i, j), if either xi or yj is greater than 0, calculate:
  - If xi = yj, the value is the diagonal cell + 1.
  - Otherwise, take the maximum from the cell directly above or to the left.
- The bottom-right cell contains the length of the LCS.
### LCS: Bottom-up DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td><strong>j</strong></td>
<td>y_j</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>0</td>
<td>x_i</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>←</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>←</td>
<td>1</td>
<td>←</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>2</td>
<td>←</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>0</td>
<td>↑</td>
<td>↖</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>↖</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>0</td>
<td>↖</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>4</td>
</tr>
</tbody>
</table>
PRINT-LCS ( b, X, i, j )

1. if i = 0 or j = 0
2. return
3. if b[i,j] = “\n”
4. PRINT-LCS ( b,X,i − 1,j − 1 )
5. print xi
6. elseif b[i,j] = “↑”
7. PRINT-LCS ( b,X,i − 1,j )
8. else PRINT-LCS ( b,X,i,j − 1 )

Running time = O(m + n)
LCS: Linear Space with Recursive Divide-&-Conquer

\[ Q \equiv c[1 \ldots n, 1 \ldots n] \]

\[ n = 2^q \]
LCS: Linear Space with Recursive Divide-&-Conquer

1. Decompose Q:
   Split Q into four quadrants.

2. Forward Pass (Generate Boundaries):
   Generate the right and the bottom boundaries of the quadrants recursively.
   (of at most 3 quadrants)
1. **Decompose Q:**
   Split $Q$ into four quadrants.

2. **Forward Pass (Generate Boundaries):**
   Generate the right and the bottom boundaries of the quadrants recursively.
   (of at most 3 quadrants)

3. **Backward Pass (Extract LCS-Path Fragments):**
   Extract LCS-Path fragments from the quadrants recursively.
   (from at most 3 quadrants)

4. **Compose LCS-Path:**
   Combine the LCS-Path fragments.
Optimal Binary Search Trees (OPBST)

Given (1) a sequence \( K = \langle k_1, k_2, \ldots, k_n \rangle \) of \( n \) distinct key in sorted order (so that \( k_1 < k_2 < \cdots < k_n \)),

(2) for \( i \in [1, n] \), probability \( p_i \) that a search will be for \( k_i \),

(3) for \( i \in [1, n-1] \), probability \( q_i \) that a search will be for a key (say, \( d_i \)) between \( k_i \) and \( k_{i+1} \),

(4) probability \( q_0 \) that a search will be for a key (say, \( d_0 \)) smaller than \( k_1 \), and

(5) probability \( q_n \) that a search will be for a key (say, \( d_n \)) larger than \( k_n \).

So, \( \sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1 \)

Construct a binary search tree \( T \) from keys in \( K \) such that the following expected search cost in \( T \) is minimized:

\[
\sum_{i=1}^{n} (\text{depth}(k_i) + 1). p_i + \sum_{i=0}^{n} (\text{depth}(d_i) + 1). q_i
\]
Optimal Binary Search Trees (OPBST)

<table>
<thead>
<tr>
<th>$k_i$</th>
<th>$k_1$</th>
<th>$k_2$</th>
<th>$k_3$</th>
<th>$k_4$</th>
<th>$k_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_i$</td>
<td>0.15</td>
<td>0.10</td>
<td>0.05</td>
<td>0.10</td>
<td>0.20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$d_i$</th>
<th>$d_0$</th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_i$</td>
<td>0.05</td>
<td>0.10</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
<td>0.10</td>
</tr>
</tbody>
</table>
Optimal Binary Search Trees (OPBST)

<table>
<thead>
<tr>
<th>node</th>
<th>depth</th>
<th>probability</th>
<th>contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>1</td>
<td>0.15</td>
<td>0.30</td>
</tr>
<tr>
<td>$k_2$</td>
<td>0</td>
<td>0.10</td>
<td>0.10</td>
</tr>
<tr>
<td>$k_3$</td>
<td>2</td>
<td>0.05</td>
<td>0.15</td>
</tr>
<tr>
<td>$k_4$</td>
<td>1</td>
<td>0.10</td>
<td>0.20</td>
</tr>
<tr>
<td>$k_5$</td>
<td>2</td>
<td>0.20</td>
<td>0.60</td>
</tr>
<tr>
<td>$d_0$</td>
<td>2</td>
<td>0.05</td>
<td>0.15</td>
</tr>
<tr>
<td>$d_1$</td>
<td>2</td>
<td>0.10</td>
<td>0.30</td>
</tr>
<tr>
<td>$d_2$</td>
<td>3</td>
<td>0.05</td>
<td>0.20</td>
</tr>
<tr>
<td>$d_3$</td>
<td>3</td>
<td>0.05</td>
<td>0.20</td>
</tr>
<tr>
<td>$d_4$</td>
<td>3</td>
<td>0.05</td>
<td>0.20</td>
</tr>
<tr>
<td>$d_5$</td>
<td>3</td>
<td>0.10</td>
<td>0.40</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>2.80</td>
<td></td>
</tr>
</tbody>
</table>
Optimal Binary Search Trees (OPBST)

<table>
<thead>
<tr>
<th>node</th>
<th>depth</th>
<th>probability</th>
<th>contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>1</td>
<td>0.15</td>
<td>0.30</td>
</tr>
<tr>
<td>$k_2$</td>
<td>0</td>
<td>0.10</td>
<td>0.10</td>
</tr>
<tr>
<td>$k_3$</td>
<td>3</td>
<td>0.05</td>
<td>0.20</td>
</tr>
<tr>
<td>$k_4$</td>
<td>2</td>
<td>0.10</td>
<td>0.30</td>
</tr>
<tr>
<td>$k_5$</td>
<td>1</td>
<td>0.20</td>
<td>0.40</td>
</tr>
<tr>
<td>$d_0$</td>
<td>2</td>
<td>0.05</td>
<td>0.15</td>
</tr>
<tr>
<td>$d_1$</td>
<td>2</td>
<td>0.10</td>
<td>0.30</td>
</tr>
<tr>
<td>$d_2$</td>
<td>4</td>
<td>0.05</td>
<td>0.25</td>
</tr>
<tr>
<td>$d_3$</td>
<td>4</td>
<td>0.05</td>
<td>0.25</td>
</tr>
<tr>
<td>$d_4$</td>
<td>3</td>
<td>0.05</td>
<td>0.20</td>
</tr>
<tr>
<td>$d_5$</td>
<td>2</td>
<td>0.10</td>
<td>0.30</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>2.75</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>depth</th>
<th>probability</th>
<th>contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>1</td>
<td>0.15</td>
<td>0.30</td>
</tr>
<tr>
<td>$k_2$</td>
<td>0</td>
<td>0.10</td>
<td>0.10</td>
</tr>
<tr>
<td>$k_3$</td>
<td>3</td>
<td>0.05</td>
<td>0.20</td>
</tr>
<tr>
<td>$k_4$</td>
<td>2</td>
<td>0.10</td>
<td>0.30</td>
</tr>
<tr>
<td>$k_5$</td>
<td>1</td>
<td>0.20</td>
<td>0.40</td>
</tr>
<tr>
<td>$d_0$</td>
<td>2</td>
<td>0.05</td>
<td>0.15</td>
</tr>
<tr>
<td>$d_1$</td>
<td>2</td>
<td>0.10</td>
<td>0.30</td>
</tr>
<tr>
<td>$d_2$</td>
<td>4</td>
<td>0.05</td>
<td>0.25</td>
</tr>
<tr>
<td>$d_3$</td>
<td>4</td>
<td>0.05</td>
<td>0.25</td>
</tr>
<tr>
<td>$d_4$</td>
<td>3</td>
<td>0.05</td>
<td>0.20</td>
</tr>
<tr>
<td>$d_5$</td>
<td>2</td>
<td>0.10</td>
<td>0.30</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>2.75</td>
<td></td>
</tr>
</tbody>
</table>
OPBST: Recurrence

Let \( w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l \) for \( 1 \leq i \leq j \leq n \).

Let \( e(i, j) \) = expected cost of searching an optimal binary search tree containing the keys \( k_i, ..., k_j \).

Then \( e(1, n) \) = expected cost of searching an optimal binary search tree containing \( k_1, ..., k_n \) (i.e., containing all keys).

If \( k_r \) is the root of an optimal subtree containing \( k_i, ..., k_j \), then

\[
e(i, j) = p_r + \left\{ e(i, r - 1) + w(i, r - 1) \right\} \\
+ \left\{ e(r + 1, j) + w(r + 1, j) \right\} \\
= e(i, r - 1) + e(r + 1, j) + w(i, j)
\]

Hence,

\[
e(i, j) = \begin{cases} 
q_{i-1}, & \text{if } j = i - 1, \\
\min_{i \leq r \leq j} \left\{ e(i, r - 1) + e(r + 1, j) + w(i, j) \right\}, & \text{if } i < j.
\end{cases}
\]
**OPBST: Bottom-up DP (Cubic Time)**

OPTIMAL-BST (p, q, n)

1. \( e[1..n+1,0..n] \leftarrow \text{new table}, \)
   \( w[1..n+1,0..n] \leftarrow \text{new table}, \)
   \( \text{root}[1..n,1..n] \leftarrow \text{new table} \)

2. \( \text{for } i \leftarrow 1 \text{ to } n+1 \text{ do} \)
3. \( e[i,i-1] \leftarrow q_{i-1} \)
4. \( w[i,i-1] \leftarrow q_{i-1} \)
5. \( \text{for } l \leftarrow 1 \text{ to } n \text{ do} \)
6. \( \quad \text{for } i \leftarrow 1 \text{ to } n-l+1 \text{ do} \)
7. \( j \leftarrow i + l - 1 \)
8. \( e[i,i] \leftarrow \infty \)
9. \( w[i,j] \leftarrow w[i,j-1] + p_{j} + q_{j} \)
10. \( \quad \text{for } r \leftarrow i \text{ to } j \text{ do} \)
11. \( t \leftarrow e[i,r-1] + e[r+1,j] + w[i,j] \)
12. \( \quad \text{if } t < e[i,j] \text{ then} \)
13. \( e[i,j] \leftarrow t \)
14. \( \quad \text{root}[i,j] \leftarrow r \)
15. \( \text{return } e \text{ and root} \)

Running time = \( \Theta(n^3) \)
OPBST: Bottom-up DP (Cubic Time)
**OPTIMAL-BST (p, q, n)**

1. $e[1..n + 1, 0..n] \leftarrow$ new table,
   $w[1..n + 1, 0..n] \leftarrow$ new table,
   $root[1..n, 1..n] \leftarrow$ new table
2. for $i \leftarrow 1$ to $n + 1$ do
3. \hspace{1em} $e[i, i - 1] \leftarrow q_{i-1}$
4. \hspace{1em} $w[i, i - 1] \leftarrow q_{i-1}$
5. for $l \leftarrow 1$ to $n$ do
6. \hspace{1em} for $i \leftarrow 1$ to $n - l + 1$ do
7. \hspace{2em} $j \leftarrow i + l - 1$
8. \hspace{2em} $e[i, j] \leftarrow \infty$
9. \hspace{2em} $w[i, j] \leftarrow w[i, j - 1] + p_j + q_j$
10. \hspace{2em} for $r \leftarrow root[i, j - 1]$ to $root[i + 1, j]$ do
11. \hspace{3em} $t \leftarrow e[i, r - 1] + e[r + 1, j] + w[i, j]$
12. \hspace{3em} if $t < e[i, j]$ then
13. \hspace{4em} $e[i, j] \leftarrow t$
14. \hspace{4em} $root[i, j] \leftarrow r$
15. return $e$ and $root$

Running time $= \Theta(n^2)$
An *Increasing Subsequence* $L$ of a given sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$ of numbers is obtained by deleting zero or more numbers from $A$ such that every number $x \in L$ is larger than the number immediately preceding $x$ in $L$.

A *Longest Increasing Subsequence (LIS)* of $A$ has the maximum length among all increasing subsequences of $A$. 
Let’s augment the given sequence $A = \langle a_1, a_2, ..., a_n \rangle$ to include a sentinel value $a_0 = -\infty$. Thus $\langle a_0, a_1, a_2, ..., a_n \rangle$ is our augmented sequence.

Let $LIS(i)$ be the length of the longest increasing subsequence of $\langle a_i, a_{i+1}, ..., a_n \rangle$ that starts at $a_i$.

Then

$$LIS(i) = 1 + \max_{i < j \leq n} \{ LIS(j) \mid a_j > a_i \}$$

Running time $= \Theta(n^2)$. 

**Longest Increasing Subsequence (LIS)**
Subset Sum

Given an array $A[1..n]$ of $n$ positive integers and a target integer $T$, determine if any subset of the numbers in $A$ sum up to $T$. 
Subset Sum

Given an array $A[1..n]$ of $n$ positive integers and a target integer $T$, determine if any subset of the numbers in $A$ sum up to $T$. Let $S(i, t)$ be True iff some subset of $A[i..n]$ adds up to $t$.

Then

$$S(i, t) = \begin{cases} 
  True, & \text{if } t = 0, \\
  False, & \text{if } t < 0 \text{ or } i > n, \\
  S(i + 1, t) \lor S(i + 1, t - A[i]), & \text{otherwise}.
\end{cases}$$

Running time $= \Theta(nT)$.

The resulting DP algorithm is called a pseudo-polynomial time algorithm because its running time depends on the numeric value of the input.
The Knapsack Problem

You have a knapsack of integer weight capacity $W$.

There are $n$ items to pick from with the $i^{th}$ item having weight $w_i$ and value $v_i$, where $1 \leq i \leq n$. All weight values are integers.

You need to pickup the most valuable combination of items that fit in your knapsack

**Unbounded Knapsack:**
Pick up as many copies of each item as you want.

**0/1 Knapsack:**
Pick up at most one copy of each item.
The Knapsack Problem

You have a knapsack of integer weight capacity $W$.

There are $n$ items to pick from with the $i^{th}$ item having weight $w_i$ and value $v_i$, where $1 \leq i \leq n$. All weight values are integers.

You need to pickup the most valuable combination of items that fit in your knapsack

**Unbounded Knapsack:**

Pick up as many copies of each item as you want.

Let $K(w) = \text{maximum value achievable with a knapsack of capacity } w$.

Then $K(w) = \max_{i: w_i \leq w} \{K(w - w_i) + v_i\}$

Running time $= \Theta(nW)$. 
The Knapsack Problem

You have a knapsack of integer weight capacity $W$.

There are $n$ items to pick from with the $i^{th}$ item having weight $w_i$ and value $v_i$, where $1 \leq i \leq n$. All weight values are integers.

You need to pickup the most valuable combination of items that fit in your knapsack

0/1 Knapsack:

Pick up at most one copy of each item.

Let $K(w, i) = $ maximum value achievable with a knapsack of capacity $w$ and items 1,2, ..., $i$.

Then $K(w, i) = \max\{K(w - w_i, i - 1) + v_i, K(w, i - 1)\}$

Running time = $\Theta(nW)$. 