Lecture 16
Dynamic Programming
• Longest Common Subsequence

Adam Smith

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Least Common Subsequence

A.k.a. “sequence alignment”
“edit distance”

…
Longest Common Subsequence (LCS)

• Given two sequences \( x[1 \ldots m] \) and \( y[1 \ldots n] \), find a longest subsequence common to them both.

“a” not “the”

\[
x: \quad A \quad B \quad C \quad B \quad D \quad A \quad B
\]
\[
y: \quad B \quad D \quad C \quad A \quad B \quad A
\]

\[
\text{BCBA} = \text{LCS}(x, y)
\]
Longest Common Subsequence (LCS)

- Given two sequences $x[1 \ldots m]$ and $y[1 \ldots n]$, find a longest subsequence common to them both.

"a" not "the"

$x: \begin{array}{cccccc} A & B & C & B & D & A & B \\ \end{array}$

$y: \begin{array}{cccccc} B & D & C & A & B & A \\ \end{array}$

$LCS(x, y) = \begin{array}{cccccc} BCAB \end{array}$
Motivation

- Approximate string matching [Levenshtein, 1966]
  - Search for “occurance”, get results for “occurrence”
- Computational biology [Needleman-Wunsch, 1970’s]
  - Simple measure of genome similarity
    
    cgtacgtacgtacgtacgtacgtacgtacgtatcgtacgt
    acgtacgtacgtacgtacgtacgtacgtacgtacgtacgt
Motivation

• Approximate string matching [Levenshtein, 1966]
  – Search for “occurance”, get results for “occurrence”
• Computational biology [Needleman-Wunsch, 1970’s]
  – Simple measure of genome similarity
    \[
    \text{acgtacgtacgtacgtacgtacgtatcgta} \quad \text{tcgtacgttacgtacgt}
    \text{aacgtacgtacgtacgtacgtacgta} \quad \text{tcgtacgt}
    \]

• \( n - \text{length}(\text{LCS}(x,y)) \) is called the “edit distance”
Brute-force LCS algorithm

Check every subsequence of $x[1 \ldots m]$ to see if it is also a subsequence of $y[1 \ldots n]$.

**Analysis**

- Checking = $O(n)$ time per subsequence.
- $2^m$ subsequences of $x$ (each bit-vector of length $m$ determines a distinct subsequence of $x$).

Worst-case running time = $O(n2^m)$

= exponential time.
Dynamic programming algorithm

**Simplification:**
1. Look at the *length* of a longest-common subsequence.
2. Extend the algorithm to find the LCS itself.

**Notation:** Denote the length of a sequence $s$ by $|s|$.

**Strategy:** Consider *prefixes* of $x$ and $y$.
- Define $c[i, j] = |\text{LCS}(x[1 \ldots i], y[1 \ldots j])|$. 
- Then, $c[m, n] = |\text{LCS}(x, y)|$. 
Recursive formulation

\[ c[i, j] = \begin{cases} 
    c[i-1, j-1] + 1 & \text{if } x[i] = y[j], \\
    \max\{c[i-1, j], c[i, j-1]\} & \text{otherwise.}
\end{cases} \]

Base case: \( c[i,j] = 0 \) if \( i=0 \) or \( j=0 \).

Case \( x[i] = y[j] \):

The second case is similar.
Dynamic-programming hallmark

#1

**Optimal substructure**
An optimal solution to a problem (instance) contains optimal solutions to subproblems.

If $z = \text{LCS}(x, y)$, then any prefix of $z$ is an LCS of a prefix of $x$ and a prefix of $y$. 

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Recursive algorithm for LCS

\[
\text{LCS}(x, y, i, j) \quad // \text{ignoring base cases}
\]

\[
\text{if } x[i] = y[j] \\
\quad \text{then } c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1 \\
\quad \text{else } c[i, j] \leftarrow \max \{ \text{LCS}(x, y, i-1, j), \text{LCS}(x, y, i, j-1) \}
\]

\text{return } c[i, j]

\textbf{Worse case: } x[i] \neq y[j], \text{ in which case the algorithm evaluates two subproblems, each with only one parameter decremented.}
Recursion tree

$m = 7, n = 6$:

$7,6$

$6,6$

$5,6$

$4,6$

$6,5$

$5,5$

$5,5$

$6,4$

$6,4$

$6,5$

$6,5$

$5,5$

$6,4$

$6,4$

$7,4$

$7,3$

Height $= m + n \Rightarrow$ work potentially exponential.

2/20/12

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Recursion tree

\[ m = 7, \ n = 6: \]

Height = \( m + n \) \(\Rightarrow\) work potentially exponential, but we’re solving subproblems already solved!

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Dynamic-programming hallmark #2

**Overlapping subproblems**

A recursive solution contains a “small” number of distinct subproblems repeated many times.

The number of distinct LCS subproblems for two strings of lengths $m$ and $n$ is only $mn$. 
Memoization: After computing a solution to a subproblem, store it in a table. Subsequent calls check the table to avoid redoing work.

\[ \text{LCS}(x, y, i, j) \]

- if \( c[i,j] = \text{NIL} \)
  - then if \( x[i] = y[j] \)
    - then \( c[i,j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1 \)
  - else \( c[i,j] \leftarrow \max \{ \text{LCS}(x, y, i-1, j), \text{LCS}(x, y, i, j-1) \} \)

\[ \text{Time} = \Theta(mn) = \text{constant work per table entry.} \]
\[ \text{Space} = \Theta(mn). \]
**Dynamic-programming algorithm**

**Idea:**
Compute the table bottom-up.

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>B</th>
<th>D</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</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>B</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></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></td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
```

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Dynamic-programming algorithm

**Idea:**
Compute the table bottom-up.

Time $= \Theta(mn)$. 

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>B</th>
<th>D</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</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>B</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B</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>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
**Dynamic-programming algorithm**

**Idea:**
Compute the table bottom-up.

Time = $\Theta(mn)$.

Reconstruct LCS by tracing backwards.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>B</th>
<th>D</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</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>B</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</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>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</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>D</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

*2/20/12*

*A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne*
**IDEA:**

Compute the table bottom-up.

Time $= \Theta(mn)$.

Reconstruct LCS by tracing backwards.

Multiple solutions are possible.

---

2/20/12

*A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne*
Dynamic-programming algorithm

**Idea:**
Compute the table bottom-up.

Time $= \Theta(mn)$.

Reconstruct LCS by tracing backwards.

Space $= \Theta(mn)$.

With tweaks:
Space $O(\min\{m, n\})$.