CSE 421
Algorithms
Richard Anderson
Lecture 19
Longest Common Subsequence

Longest Common Subsequence
• $C = c_1 \ldots c_g$ is a subsequence of $A = a_1 \ldots a_m$ if $C$ can be obtained by removing elements from $A$ (but retaining order)
• $LCS(A,B)$: A maximum length sequence that is a subsequence of both $A$ and $B$

Determine the LCS of the following strings

BARTHOLEMEWSIMPSON
KRUSTYTHECLOWN

String Alignment Problem
• Align sequences with gaps
  
  \[
  \begin{array}{cccc}
  CAT & TGA & AT & \\
  CAGAT & AGGA & \\
  \end{array}
  \]
• Charge $\delta_x$ if character $x$ is unmatched
• Charge $\gamma_{xy}$ if character $x$ is matched to character $y$

Note: the problem is often expressed as a minimization problem, with $\gamma_{xx} = 0$ and $\delta_x > 0$

LCS Optimization
• $A = a_1a_2\ldots a_m$
• $B = b_1b_2\ldots b_n$
• $Opt[j,k]$ is the length of $LCS(a_1a_2\ldots a_j, b_1b_2\ldots b_k)$

Optimization recurrence
If $a_j = b_k$, $Opt[j,k] = 1 + Opt[j-1, k-1]$
If $a_j \neq b_k$, $Opt[j,k] = \max(\text{Opt}[j-1,k], \text{Opt}[j,k-1])$
Give the Optimization Recurrence for the String Alignment Problem

- Charge $\delta_x$ if character $x$ is unmatched
- Charge $\gamma_{xy}$ if character $x$ is matched to character $y$

$$\text{Opt}[j, k] = \begin{cases} \text{Let } a_i = x \text{ and } b_k = y \\ \text{Express as minimization} \end{cases}$$

Dynamic Programming Computation

Code to compute $\text{Opt}[j, k]$

Storing the path information

How good is this algorithm?

- Is it feasible to compute the LCS of two strings of length 100,000 on a standard desktop PC? Why or why not.

Observations about the Algorithm

- The computation can be done in $O(m+n)$ space if we only need one column of the Opt values or Best Values
- The algorithm can be run from either end of the strings
Computing LCS in $O(nm)$ time and $O(n+m)$ space

- Divide and conquer algorithm
- Recomputing values used to save space

**Divide and Conquer Algorithm**

- Where does the best path cross the middle column?
- For a fixed $i$, and for each $j$, compute the LCS that has $a_i$ matched with $b_j$

**Constrained LCS**

- $\text{LCS}_{i,j}(A,B)$: The LCS such that
  - $a_1,\ldots,a_i$ paired with elements of $b_1,\ldots,b_j$
  - $a_{i+1},\ldots,a_m$ paired with elements of $b_{j+1},\ldots,b_n$
- $\text{LCS}_{4,3}(abbacbb, cbbaa)$

**A = RRSSRTTRRTS**  
**B=RTSRRSTST**

Compute $\text{LCS}_{5,0}(A,B)$, $\text{LCS}_{5,1}(A,B)$, $\ldots$, $\text{LCS}_{5,9}(A,B)$

<table>
<thead>
<tr>
<th>j</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

**Computing the middle column**

- From the left, compute $\text{LCS}(a_1\ldots a_{m/2}, b_1\ldots b_i)$
- From the right, compute $\text{LCS}(a_{m/2+1}\ldots a_m, b_{j+1}\ldots b_n)$
- Add values for corresponding $j$’s
- Note – this is space efficient
Divide and Conquer

- $A = a_1, \ldots, a_m$, $B = b_1, \ldots, b_n$
- Find $j$ such that
  - $\text{LCS}(a_1 \ldots a_{m/2}, b_1 \ldots b_j)$ and
  - $\text{LCS}(a_{m/2+1} \ldots a_m, b_{j+1} \ldots b_n)$ yield optimal solution
- Recurse

Algorithm Analysis

- $T(m, n) = T(m/2, j) + T(m/2, n-j) + cnm$

Prove by induction that $T(m, n) \leq 2cmn$

Memory Efficient LCS Summary

- We can afford $O(nm)$ time, but we can’t afford $O(nm)$ space
- If we only want to compute the length of the LCS, we can easily reduce space to $O(n+m)$
- Avoid storing the value by recomputing values
  - Divide and conquer used to reduce problem sizes