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)
- \(\text{LCS}(A, B)\): A maximum length sequence that is a subsequence of both \(A\) and \(B\)

Instructor Example

\[
\begin{align*}
\text{occuranec} & \quad \text{attacggct} \\
\text{occurrence} & \quad \text{tacgacca}
\end{align*}
\]

Determine the LCS of the following strings

- BARTHOLEMEWSIMPSON
- KRUSTYTHECLOWN

String Alignment Problem

- Align sequences with gaps
  
  \[
  \begin{align*}
  \text{CAT} & \quad \text{TGA} & \quad \text{AT} \\
  \text{CAGAT} & \quad \text{AGGA}
  \end{align*}
  \]

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

LCS Optimization

- \(A = a_1a_2\ldots a_m\)
- \(B = b_1b_2\ldots b_n\)

- \(\text{Opt}[j, k]\) is the length of \(\text{LCS}(a_1a_2\ldots a_j, b_1b_2\ldots b_k)\)

Optimization recurrence

If \(a_j = b_k\), \(\text{Opt}[j,k] = 1 + \text{Opt}[j-1, k-1]\)

If \(a_j \neq b_k\), \(\text{Opt}[j,k] = \max(\text{Opt}[j-1,k], \text{Opt}[j,k-1])\)
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 $y$

Dynamic Programming Computation

Write the code to compute Opt[$j,k$]

```
for i := 1 to m
    Opt[i, 0] := 0;
for j := 1 to n
    Opt[0, j] := 0;
Opt[0, 0] := 0;
for i := 1 to m
    for j := 1 to n
        else if Opt[i-1, j] >= Opt[i, j-1]
            {  Opt[i, j] := Opt[i-1, j], Best[i,j] := Left; }
        else     {  Opt[i, j] := Opt[i, j-1], Best[i,j] := Down; }
```

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
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 $b_{j+1},\ldots,b_n$

• $\text{LCS}_{4,3}(abbacb, cbbab)$

A = RRSSRTTRTS
B = RTSRRSTST

Compute LCS$_{5,1}(A,B)$, LCS$_{5,2}(A,B)$, …, LCS$_{5,9}(A,B)$

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

Instructor Example

<table>
<thead>
<tr>
<th>j</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</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>3</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>

Student Submission

Computing the middle column

• From the left, compute $\text{LCS}(a_1\ldots a_{m/2},b_1\ldots b_j)$
• 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
Algorithm Analysis

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

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