**LCS Review: Lecture 15**

Let \( c_{ij} = \text{length of LCS of } x_1 x_2 ... x_i \text{ and } y_1 y_2 ... y_j \).

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

\[
\begin{align*}
b[i,j] &= \begin{cases} 
"\downarrow" & \text{if } x_i = y_j, \\
"\uparrow" & \text{if } x_i \neq y_j \text{ and } c[i-1,j-1] \geq c[i,j-1], \\
"\leftarrow" & \text{if } x_i \neq y_j \text{ and } c[i-1,j-1] < c[i,j-1].
\end{cases} \\
\end{align*}
\]

We compute the \( c[i,j] \) and \( b[i,j] \) in order of increasing \( i+j \), or alternatively in order of increasing \( i \), and for a fixed \( i \), in order of increasing \( j \).

If characters match, add 1 to value in diagonal, point to that entry.
If characters DON'T match, look to cell above and to left and enter that value in current cell. Point to larger cell entry, **default to up if equal.**
## LCS Example: Lecture 15

<table>
<thead>
<tr>
<th></th>
<th>j 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>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>x</td>
<td>y</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>0</td>
<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>b</td>
<td>10</td>
<td>10</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>2</td>
<td>a</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>d</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>b</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

http://lcs-demo.sourceforge.net/

## Edit Distance Review: Lecture 16

- Initialize matrix $d$:
  - $d(i, 0) = i$, $d(0, j) = j$ // $i = \text{row}$, $j = \text{column}$

Compute values in a similar row-centric way using a nested-loop the values:

$$d[i, j] = \begin{cases} 
\begin{align*}
&d[i-1, j-1] \quad \text{if } x_i = y_i \\
&d[i-1, j] + 1 \quad \text{if } x_i \neq y_i
\end{align*} \\
\min & \begin{align*}
&d[i, j-1] + 1 \quad \text{if } x_i \neq y_i
\end{align*}
\end{cases}$$

Arrows point to the cell used to compute the current cell's value (no default on ties, not unique!)
Example: Edit Distance, Lecture 16

Levenshtein Distance Calculator

Calculate Levenshtein distance

<table>
<thead>
<tr>
<th>ATCGTT</th>
<th>LEFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATCGTT</td>
<td>LEFT</td>
</tr>
<tr>
<td>ATCGTT</td>
<td>C</td>
</tr>
<tr>
<td>ATCGTT</td>
<td>AC</td>
</tr>
<tr>
<td>ATCGT</td>
<td>TAC</td>
</tr>
<tr>
<td>ATCG</td>
<td>TTAC</td>
</tr>
<tr>
<td>AT</td>
<td>GTTAC</td>
</tr>
<tr>
<td>A</td>
<td>GTTAC</td>
</tr>
</tbody>
</table>

The Levenshtein distance is 4: delete t at position 2, delete e at position 3, insert a at position 6, and insert c at position 6.

http://leojiang.com/experiments/levenshtein/

Edit Distance: Review

- Edit Distance:
  - Gave the minimum number of changes to convert one string into another
  - Can be used to illustrate how to transform one sequence into another
  - Can also be used for basic sequence alignment, but doesn't allow for a full cost analysis and comparison
Sequence Alignment: General

- Needleman-Wunsch
- Maximizes similarity by giving weights to the types of differences (gaps and mismatches)
- Generalized form of Levenshtein
  - additional parameters: (discipline or app. specific)
    - gap penalty, $\delta$
    - mismatch cost $\alpha_{x,y} = \text{cost of mismatched } x, y$
      match reward: $\alpha_{x,x} = 0$ (our assumption)

Sequence Alignment: Example

- Align mean and man:
  
<table>
<thead>
<tr>
<th>Option 1:</th>
<th>Option 2:</th>
<th>Option 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>mean</td>
<td>mean</td>
</tr>
<tr>
<td>man−</td>
<td>−man</td>
<td>m−an</td>
</tr>
<tr>
<td>2 mismatches</td>
<td>1 mismatch</td>
<td>0 mismatch</td>
</tr>
<tr>
<td>1 gap</td>
<td>1 gap</td>
<td>1 gap</td>
</tr>
</tbody>
</table>

Option 3 best in this case
Options 1 and 2: What if $e\rightarrow a$ mismatch (opt 1) is more expensive than $e\rightarrow m$ mismatch (opt 2)?
Sequence Alignment: Definition

- \( S_X = \{1, 2, \ldots, m\} \): positions of characters in string \( X \)
- \( S_Y = \{1, 2, \ldots, n\} \): positions of characters in string \( Y \)
- A **alignment** \( M \) is a set of pairs of positions:
  \[ M = \{(1, 2), (2, 3), (4, 4), \ldots \} \]

  such that:
  - each item in \( S_X \) and \( S_Y \) appear in at most one pair.
  - no crossings are allowed:
    - i.e. if \((i, j)\) and \((i', j')\) are both in \( M \) and \( i < i' \), then \( j < j' \).

In previous example:
- Option 3: \( M = \{(1, 1), (3, 2), (4, 3)\} \)

Sequence Alignment: Cost

- The cost of a sequence alignment \( M \) is the sum of:
  - Costs of all of the mismatches in \( M \)
  - Costs of all of the gaps in either string:

\[
Cost(M) = \sum \alpha_{x_i,y_j} + \sum \delta_{\text{gaps in } x} + \sum \delta_{\text{gaps in } y}
\]

In option 1 of our example, the cost is:

\[
Cost(M) = \alpha_{e,a} + \alpha_{a,n} + \delta
\]
Recurrence

- Two strings \(x_1\ldots x_m\) and \(y_1\ldots y_n\)
- In an **optimal** alignment \(M\), at least one of the following is true:
  - (i) \((m, n)\) is in \(M\) // last characters are matched
  - (ii) \(x_m\) is not matched // No pair \((m, *)\) in \(M\)
  - (iii) \(y_n\) is not matched // No pair \((*, n)\) in \(M\)

Recurrence: Consequence

Let \(\text{opt}(i, j)\) be the **minimum cost** of aligning
\[X_i = \{x_1, x_2, \ldots x_i\}\] and \(Y_j = \{y_1, y_2, \ldots y_j\}\)

The cases from previous slide:
- (i) pay mismatch cost \(\alpha_{x_i, y_j}\) and align \(X_{i-1}\) and \(Y_{j-1}\)
- (ii) pay gap cost \(\delta\) since \(x_i\) is not matched then align \(X_{i-1}\) and \(Y_j\)
- (iii) pay gap cost \(\delta\) since \(y_j\) is not matched then align \(X_i\) and \(Y_{j-1}\)

Take the minimum of these three costs!
Recurrence**

So, for $i$ and $j > 0$:

$\text{opt}(i, j) = \min[\alpha_{x_i,y_j} + \text{opt}(i-1, j-1), (\text{DIAG})$
$\delta + \text{opt}(i-1, j), \text{ // } x_i \text{ is not matched (UP)}$
$\delta + \text{opt}(i, j-1), \text{ // } y_j \text{ is not matched (L)}$

$(x_i, y_j)$ is in an optimal alignment $M$ for this subproblem iff the minimum achieved is achieved by the first of these three values.

Sequence Alignment

Alignment($X, Y$)  // $|X| = m$, $|Y| = n$
Array $A[0...m, 0...n]$
$A[i,0]=i*\delta$  // must use all gaps
$A[0,j]=j*\delta$  // to align these
For $i = 1$ to $m$  // row centric
    For $j = 1$ to $n$
        use recurrence ** to compute $A[i,j]$
    Endfor
Endfor
Return $A[m,n]$
Example

• Assume that:
  • $\delta = 2$
  • $\alpha (v, v) = 1$
  • $\alpha (c, c) = 1$
  • $\alpha (v, c) = 3$
• What is the cost of aligning the strings?:
  • man
  • mean

Example, Continued

<table>
<thead>
<tr>
<th></th>
<th>m</th>
<th>e</th>
<th>a</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\delta = 2$
$\alpha (v, v) = 1$
$\alpha (c, c) = 1$
$\alpha (v, c) = 3$

$\text{opt}(i, j) = \min\{\alpha_{x_i,y_j} + \text{opt}(i - 1, j - 1), \text{(DIAG)}\}$
$\delta + \text{opt}(i - 1, j), \quad \|x_i\text{ is not matched (UP)}$
$\delta + \text{opt}(i, j - 1), \quad \|y_j\text{ is not matched (L)}$
Recover Alignment

\[ \delta = 2 \]
\[ \alpha (v, v) = 1 \]
\[ \alpha (c, c) = 1 \]
\[ \alpha (v, c) = 3 \]

\textbf{Figure 6.18} The opt values for the problem of aligning the words \textit{mean} to \textit{name}.

Kleinberg, Tardos, p 284

Actual Alignment

- How do we recover the actual alignment?

- Do we need the entire matrix?
Time, Space Complexity

- Standard Algorithm:
  - Time: $O(mn)$
  - Space: $O(mn)$
  - Upside: Recovering actual alignment relatively easy

- Space-Efficient Algorithm:
  - Time: $O(mn)$
  - Space: $O(m+n)$  //For recovering alignment
  - Downside: Makes recovering actual alignment more difficult

Space complexity is important in this context!

Sequence Alignment (space efficient)

- Hirschberg – 1975
- value: need the current and previous column

```plaintext
Space-Efficient-Alignment(X, Y)

Array B[0...m, 0...1]
Initialize B[i, 0] \equiv \delta for each i (just as in column 0 of A)
For j = 1, ..., n
    B[0, j] \equiv \delta (since this corresponds to entry A[0,j])
    For i = 1, ..., m
        B[i, j] = \min(\sigma_{xy} + B[i-1, 0],
                      \delta + B[i-1, 1],
                      \delta + B[i, 0])
    Endfor

Move column 1 of B to column 0 to make room for next iteration:
   Update B[i, 0] = B[i, 1] for each i
Endfor
```
**Algorithm**

`Divide-and-Conquer-Alignment(X,Y)`

Let \( m \) be the number of symbols in \( X \)

Let \( n \) be the number of symbols in \( Y \)

If \( m \leq 2 \) or \( n \leq 2 \) then

  Compute optimal alignment using `Alignment(X,Y)`

Call `Space-Efficient-Alignment(X,Y[1:n/2])`

Call `Backward-Space-Efficient-Alignment(X,Y[n/2+1:n])`

Let \( q \) be the index minimizing \( f(q,n/2) + g(q,n/2) \)

Add \((q,n/2)\) to global list \( P \)

`Divide-and-Conquer-Alignment(X[1:q],Y[1:n/2])`

`Divide-and-Conquer-Alignment(X[q+1:n],Y[n/2+1:n])`

Return \( P \)

---

Kleinberg, Tardos, p 288

---

**Sequence alignment: Linear Space**

---

Kleinberg, Tardos, p 289