Longest Common Subsequence (LCS)

- One measure of similarity between two DNA strands:
  - ACCCGGGTTAACKCCCGGG
  - CCGGGGTATTAAAGGCCGA

  We will study others.

- Comment from Subversion differencing engine:

```c
/*
 * Calculate the Longest Common Subsequence between two datasources.
 * This function is what makes the diff code tick.
 *
 * The LCS algorithm implemented here is described by Sun Wu.
 * Udi Manber and Gene Myers in "An O(NP) Sequence Comparison Algorithm"
 * /
```
Subsequence

- A subsequence of a string S is a set of characters that appear in left-to-right order, but not necessarily consecutively
- Example: ACTTGCG
- Subsequences:
  - ACT
  - ATTC
  - ACTTGC
- TTA is not a subsequence!

Common Subsequence

- A common subsequence of two strings is a subsequence that appears in both strings
- Example:
  - ACTTGCG and AGTCTCG
- Common subsequences
  - ATT
  - AGCG
Longest Common Subsequence

- Problem: Let \( x_1x_2...x_m \) and \( y_1y_2...y_n \) be two sequences over some alphabet.
  - We assume they are strings of characters

- Find a longest common subsequence (LCS) of \( x_1x_2...x_m \) and \( y_1y_2...y_n \)

Example

- \( x_1x_2x_3x_4x_5x_6x_7x_8 = b \ a \ c \ b \ f \ f \ c \ b \)
- \( y_1y_2y_3y_4y_5y_6y_7y_8y_9 = d \ a \ b \ e \ a \ b \ f \ b \ c \)

- Longest Common Subsequence is:

Reminder: A subsequence is a set of characters that appear in left- to-right order, but not necessarily consecutively.
Naïve Algorithm

Dynamic Programming

- LCS can be solved using dynamic programming:
  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 bottom-up
  4. Construct an optimal solution from the computed information
Step 1: Characterizing

Characterizing a longest subsequence

- **Optimal substructure**: If \( z = z_1z_2...z_p \) is a LCS of \( x_1x_2...x_m \) and \( y_1y_2...y_n \), then at least one of these most hold:
  - \( x_m = y_n \), and \( z_1z_2...z_{p-1} \) is an LCS of \( x_1x_2...x_{m-1} \) and \( y_1y_2...y_{n-1} \),
  - \( x_m \neq y_n \), and \( z_1z_2...z_p \) is an LCS of \( x_1x_2...x_{m-1} \) and \( y_1y_2...y_n \),
  - \( x_m \neq y_n \), and \( z_1z_2...z_p \) is an LCS of \( x_1x_2...x_m \) and \( y_1y_2...y_{n-1} \).

Step 2: Recursive Solution

Let \( c_p = \) length of LCS of \( x_1x_2...x_i \) and \( y = y_1y_2...y_j \).

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

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 \( j \).

C: length matrix (or cost...so maximize cost)
B: how to move around matrix (points to the optimal subproblem)
Steps 3 and 4

LCS-LENGTH(X, Y)
1 \( m = X\.\_\text{length} \)
2 \( n = Y\.\_\text{length} \)
3 let \( b[1..m, 1..n] \) and \( c[0..m, 0..n] \) be new tables
4 for \( i = 1 \) to \( m \)
5 \( c[i, 0] = 0 \)
6 for \( j = 0 \) to \( n \)
7 \( c[0,j] = 0 \)
8 for \( i = 1 \) to \( m \)
9 for \( j = 1 \) to \( n \)
10 if \( x_i == y_j \)
11 \( c[i,j] = c[i-1,j-1] + 1 \)
12 \( b[i,j] = "\backslash" \)
13 \( \text{else if } c[i-1,j] \geq c[i,j-1] \)
14 \( c[i,j] = c[i-1,j] \)
15 \( b[i,j] = "\uparrow" \)
16 \( \text{else } c[i,j] = c[i,j-1] \)
17 \( b[i,j] = "\downarrow" \)
18 return \( c \) and \( b \)

Example

b,c matrices combined

<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>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td></td>
<td></td>
<td></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>
<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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

http://lcs-demo.sourceforge.net/ is helpful step-by-step walkthrough
Printing the LCS

PRINT-LCS(b, X, i, j)
    if i == 0 or j == 0
        return
    if b[i, j] == “\”
        PRINT-LCS(b, X, i − 1, j − 1)
        print x_i
    elseif b[i, j] == “↑”
        PRINT-LCS(b, X, i − 1, j)
    else PRINT-LCS(b, X, i, j − 1)

Another Example

- What is the LCS in
  - epidemiologist
  - refrigeration
Another Example

- HUMAN versus CHIMPANZEE

http://wordaligned.org/articles/longest-common-subsequence