Longest Common Subsequence

Compare DNA of 2 organisms

$S_1 = ACCGGTCACT...$ $CCGTCT$
$S_2 = TCCGATCTG...$

Many ways to measure similarity:
- # identical characters in the same index
- # substrings in common
- longest common subsequence

First, what is a subsequence?

Informally: string $Z$ is a subsequence of string $X$ if $Z$ is made up of characters from $X$ in the same order they appear in $X$.

Formally: For a string $X = x_1x_2...x_m$, $Z$ is a length-$k$ subsequence of $X$ if there exists "an increasing sequence of indices" $i_1 < i_2 < ... < i_k$ such that $Z = x_{i_1}x_{i_2}...x_{i_k}$

Subsequences of $S_1$? A GCC CTG GA

✓ ✓ x ✓
Longest Common Subsequence (LCS) of strings $X, Y$ is longest subsequence that both $X, Y$ contain.

**LCS Problem** - Given 2 strings $X = x_1, x_2, \ldots, x_m$ and $Y = y_1, y_2, \ldots, y_n$, find the LCS of $X$ and $Y$. Denote $\text{LCS}(X, Y)$.

$LCS(S_1, S_2) = CCCCTCT$

We will start with finding the length of the LCS. Denote $|\text{LCS}(X, Y)|$.

**Brute Force?**
1. Enumerate all subsequences of $X$ and all subsequences of $Y$.
   - $2^m$ subsequences of $X$? for each character: 2 choices -
     - 2 sub-sequences of $Y$ either include it in the subsequence or don't.
   
2. Hash one list into table. Hash the other while storing collisions. (collisions indicate common subsequences)
   \(\Rightarrow 0(2^m + 2^n)\)

3. Scan collisions to find the one of max length
   Total: \(0(2^m + 2^n)\)

**Greedy?** Unclear what a greedy approach would be.

**Dynamic Programming**

Does problem exhibit optimal substructure property? Yes $\checkmark$

\[\Rightarrow \text{LCS of } X, Y \text{ depends on LCS of substrings of } X, Y.\]
Which substrings should we consider?

Suppose: \( S_1 = \overline{ACCCC...CC} \) \( \Rightarrow \) \( \text{LCS}(x, y) = A \).
\( S_2 = \overline{BBBBB...BBA} \)

- or - (vice versa)
\( S_1 = \overline{BBBBB...BBA} \)
\( S_2 = \overline{ACCCC...CC} \)

"Need to consider all of \( S_1 \) and all of \( S_2 \) starting from the beginning"

Prefix any leading (starting from beginning)
contiguous (adjacent) substring

Ex: \( X = BACDB \) prefixes: \( B, BA, BAC, BACD \) ...  
\( Y = BDEB \) prefixes: \( B, BD, BDE, \ldots \)

Denote:
\( \overline{X}_i = \text{length}-i \text{ prefix of } X \)
\( \overline{Y}_i = \text{""""""""""""""y} \)

Ex: \( \overline{X}_3 = BAC, \overline{Y}_3 = BDE \), \( \overline{X}_2 = BA, \overline{Y}_1 = B \), etc...
Another example:

\[ X = C D D D \]
\[ Y = D D D C \]
Use solutions for shorter prefixes of $X, Y$ to find solutions for longer prefixes.

But how?

Initial idea: compare same-length prefixes of $X, Y$.

New (smaller) ex: $X = BACDB$

\[ Y = BDB \]

LCS so far: \[ \uparrow \uparrow \uparrow \uparrow \rightarrow \text{but LCS = BDB not } B! \]

Notice: Any character in $X$ can match with any character in $Y$.

\[ X = \ldots D \ldots B \text{ or } X = \ldots D \ldots B \]

\[ Y = \ldots D \ldots B \]

(can't skip a character block that character might be in LCS!)

Must compare every prefix of $X$ with every prefix of $Y$.

LCS so far?

\[ X = B \ B \ B \ B \ B \ldots \ BA \ BA \ldots \ BACDB \]
\[ Y = B \ BD \ BDE \ldots BDEB \ldots B \ BD \ BD \ldots BDB \]

CS so far: \[ B \ B \ B \ B \ B \ B \ B \ BD \ BDB \]

\[ \widetilde{x_1} \widetilde{y_1} \widetilde{x_1} \widetilde{y_2} \widetilde{x_1} \widetilde{y_3} \ldots \widetilde{x_2} \widetilde{y_1} \widetilde{x_2} \widetilde{y_2} \ldots \]

Keep this ordering in mind! Smaller prefixes first!
Now...

\[ X = \boxed{BACD} B \]

Suppose we knew LCS of some prefixes of \( X, y \).

Know \( \text{LCS}(X_{i-1}, Y_{j-1}) \)

"How to use next chars in \( X, y \) to find \( \text{LCS}(X_5, Y_4) \)?"

2 cases based on next chars: either equal or not equal

Case 1: If \( x_i = y_j \):

\[ X = \boxed{X_{i-1}} B \]
\[ Y = \boxed{Y_{j-1}} B \]

Can match \( x_i + y_j \) to get a longer LCS.

\[ \text{LCS}(X_i, Y_j) = \text{LCS}(X_{i-1}, Y_{j-1}) + 1 \]

Ques: What if \( x_i \) also matches with a previous char in \( Y \)?

When would it be better to match?

Ex: \( X = \boxed{BACD} N \)
\( x_i \) with a previous char in \( Y \)?

\( X = \boxed{BACD} N \)
\( Y = \boxed{BNE} N \)

Better to match \( x_i \) (N) with a previous \( N \) in \( Y \) only if there is a future character in \( X \) that also matches.
But we will discover this in a future computation.

\[
\begin{align*}
\bar{x}_5 & \quad \text{For } (\bar{x}_6) = \text{BACDNE} \\
(\bar{y}_3) & \quad \text{BNEN} \\
\uparrow \quad \uparrow & \\
\text{LCS}_1 & = 1 \quad \text{new LCS} = 2
\end{align*}
\]

\[\text{LCS}(\bar{x}_6, \bar{y}_3) \text{ computed after LCS}(\bar{x}_5, \bar{y}_4)\]

Would match current N (\(x_5\)) with a previous N (\(y_2\)) only if there is some future char in \(x\) that can be matched with a char in \(y\) (e.g., E in the example above).

But we would discover this in a future computation - e.g., when computing \(\text{LCS}(\bar{x}_6, \bar{y}_3)\) in the example above (the matching E's).

Case 2: \(x_i \neq y_j\)

\[\begin{align*}
X & = \text{BACD} \swarrow E \\
Y & = \text{BDEB} \swarrow \swarrow \swarrow \swarrow \swarrow \\
& \quad \text{LCS won't improve (from a previously computed LCS).}
\end{align*}\]
1. LCS($X_i$, $Y_j$) may end in $X_i$.
2. LCS($X_i$, $Y_j$) may end in $Y_j$.
3. LCS($X_i$, $Y_j$) can end in both $X_i$, $Y_j$.

$X = \{B, A, C, D, E\}$
$Y = \{B, D, E, B\}$

$LCS = \{B, D, E\}$
$LCS = \{B, C, D\}$

But these have already been found in a previous computation.

4. Or, LCS ends in neither.

$X = \{B, A, C, D, K\}$
$Y = \{B, D, E, J\}$

$LCS = \{B, D, E\}$

$|LCS(X_i, Y_j)| = \max\left\{ |LCS(X_{i-1}, Y_{j-1})|, |LCS(X_{i-1}, Y_j)|, |LCS(X_i, Y_{j-1})| \right\}$

(A) $|LCS(X_{i-1}, Y_{j-1})|$
(B) $|LCS(X_{i-1}, Y_j)|$

will always be $\leq (A, B)$.
Dynamic Programming Solution

Recall: X is length m, Y is length n

1. Size of array to store values: \( \max((m+1) \times (n+1)) \).

   Compare every prefix of X to every prefix of Y.

2. What each entry holds?
   \( C[i][j] = \text{LCSS}(X_i, Y_j) \) (length).

   \( \text{if } i = 0 \text{ or } j = 0 \) \( \text{if } x_i = y_j \)

3. Dynamic programming formulation:
   \( C[i][j] = \begin{cases} 
   0 & \text{if } i = 0 \text{ or } j = 0 \\
   C[i-1][j-1] + 1 & \text{if } x_i = y_j \\
   \max(C[i-1][j], C[i][j-1]) & \text{if } x_i \neq y_j 
   \end{cases} \)

4. Base Case: \( i = 0 \text{ or } j = 0 \).

5. To fill matrix:
   \( C[0][0...n] = 0, C[0...m][0] = 0 \)

   \( \text{for } (i = 1 \text{ to } m) \ 	ext{// every prefix of } X \\
   \text{for } (j = 1 \text{ to } n) \ 	ext{// every prefix of } Y \\
   C[i][j] = \text{<based on D.P. formulation>} \)

6. Optimal value (length)? In \( C[m][n] \).

7. Run Time \( \Theta(mn) \).
How to find actual LCS?

Quick example...