The Longest Common Subsequence (LCS)

Biological applications work with DNA sequences. A strand of DNA consists of a string of molecules which are one of four possible bases: adenine (A), guanine (G), cytosine (C) and thymine (T). Thus DNA sequences can be expressed as arrays or strings over four symbols, A, C, G, T. Biologists want to compare how "close" are two DNA strands. One of the ways proposed to model closeness is to compute the longest common subsequence (LCS) of two DNA strands.

The LCS problem: Suppose we have two sequences (arrays) $X[1..n]$ and $Y[1..m]$, where each element is one of the four bases A, C, G, T.

- We say that another sequence $Z[1..k]$ is a *subsequence* of X if there exists a strictly increasing sequence of indices $i_1 < i_2 < i_3 < ... < i_k$ such that we have $X[i_1] = Z[1], X[i_2] = Z[2], ..., X[i_k] = Z[k]$.

  For example, $[A, C, D, A]$ ($i_1 = 1, i_2 = 3, i_3 = 5, i_4 = 6$) and $[A, B, B]$ ($i_1 = 1, i_2 = 2, i_3 = 4$) are subsequences of $X = [A, B, C, B, D, A, B]$.

- We say that Z is a *common subsequence* (of X, Y) if Z is a subsequence of both X and Y.

Given two sequences X and Y of size n and m respectively, come up with an algorithm that finds their longest common subsequence (LCS) [1].

Work through the questions below to come up with a DP solution for this problem.

Example: Let $X = [A, B, C, B, D, A, B]$, $Y = [B, D, C, A, B, A]$.

1. List some common subsequences of X and Y of length 2 and 3. Can you find a common subsequence of length 4? How about 5?

2. Let's start with a simpler problem: How might you write an algorithm to check if an array is a subsequence of another array ? How long?

   ```
   //return True if X is a subsequence of Y, False otherwise
   isSubsequence(array X, array Y)
   ```

[1] Also on Leetcode 1143

3. Consider a brute-force algorithm for finding $LCS(X, Y)$. For example you could enumerate all possibilities (how many possibilities are there)? You don't need to write this in details, just sketch the idea and pencil its running time.

4. **Optimal structure:** To describe optimal substructure we introduce the following notation:

   Prefixes: For any sequence $X[1..n]$ let X_i denote the sequence consisting of the first i elements of X, called the i-prefix: $X_i = X[1..i]$.

   When we think about optimal substructure we always start with an optimal solution. Let $Z[1..k] = Z_k$ be the LCS of X, Y. Let's assume we know Z_k, and let's express the optimal substructure of the problem by comparing the last elements of X and Y and Z_k and reason about Z_{k-1}.

   - Case 1: If the last elements in X and Y are equal, $X[n] == Y[m]$: Is the following claim True or False? And why?

     Claim: If $X[n] == Y[m]$ the last element of Z must be equal to the last element of X and Y, $Z[k] = X[n] = Y[m]$.
     Your answer:

     What can you say about the remaining part of Z, namely Z_{k-1}? Express it recursively in terms of X_{n-1} and Y_{m-1}:

     Your answer: $Z_{k-1} = LCS(........, ...)$

   - Case 2: If the last elements of X and Y are not equal, $X[n] \neq Y[m]$, what can you say about the last element of Z_k?
     Your answer:

     (a) Case 2 (a): If $X[n] \neq Y[m]$ and assuming someone told us that $Z[k] \neq X[n]$: What can you say about Z_k in this case? Express Z_k recursively.

     Your answer: $Z_k = LCS(......., ...)$

(b) Case 2 (b): If $X[n] \neq Y[m]$ and assuming someone told us that $Z[k] \neq Y[m]$: What can you say about Z_k in this case? Express Z_k recursively.

Your answer: $Z_k = LCS(.......,...........)$

In summary, if $X[n] \neq Y[m]$ and if someone told us if Z_k uses $X[n]$ or $Y[m]$, we knew how to recurse. But noone tells us what the last letter of Z_k is (or is not). We got to try all cases that could happen and pick the best.

5. We are now ready to write the recursive algorithm for LCS. As always, we start by computing *the length* of the LCS of X, Y. Then we'll extend the solution to compute not only the length, but the actual LCS.

   Our subproblem: c(i,j) returns the length of the LCS of X_i and Y_j.

   To find the LCS of X and Y we'll want to call $c(n, m)$.

   Start by writing the base case:

   $c(i, 0) =?$

   $c(0, j) =?$

6. If $X[i] == Y[j]$: Express $c(i, j)$ recursively.

   $c(i, j) =$

7. If $X[i] \neq Y[j]$: Express $c(i, j)$ recursively.

   $c(i, j) =$

8. Describe a DP algorithm (top-down, recursive) that $c(i, j)$.

9. How long does it take to compute $c(n, m)$ with DP?

10. Consider the following example:

$$X = [A, B, C, B, D, A, B], \quad Y = [B, D, C, A, B, A]$$

    Draw the table and show how it's filled when calling your dynamic programming function to compute $c(7, 6)$.

11. How would you extend your algorithm above to compute the LCS not just the length?

12. When you are done, step through the Python notebook to check your answer. Add code to compute the actual LCS of X, Y (not just the length).