Assignment #5
Due on Thursday 3/12/2020

Question 1 (Longest Common Subsequence) In the longest common subsequence algorithm we discussed in class, we formulated the recursive formula based on prefixes of the two inputs, i.e., \(X[1..i]\) and \(Y[1..j]\).

1. Rewrite the recursive formula using suffixes instead of prefixes, i.e., \(X[i..m]\) and \(Y[j..n]\).

2. Develop a bottom-up dynamic programming algorithm based on the recursive formula in (a). Describe the algorithm and write a pseudo code.

3. Use the algorithm developed in (b) to solve the problem for the inputs \(X = \langle A, B, C, B, D, A, B \rangle\) and \(Y = \langle B, D, C, A, B, A \rangle\). Show the final tabulation of the answer showing both the length of the answer and the arrows similar to Figure 15.8 on page 395 of the textbook (and the slides).

Question 2 (Weighted Edit Distance) Given a string \(X[1..m]\) there are three edit operations that can be applied on \(X\) to transform it to another string \(X'\). The three operations are:

1. Insert: Inserts a character to \(X\) at the \(i^{th}\) position starting at zero. For example, if \(X = \langle B, G \rangle\), then \(\text{Insert}(X, 1, 'I')\) yields the string \(X' = \langle B, I, G \rangle\). The cost of the insert operation is \(c_i\).

2. Delete: Deletes the \(i^{th}\) character from \(X\). For example, if \(X = \langle B, G \rangle\), then \(\text{Delete}(X, 0)\) yields the string \(X' = \langle G \rangle\). The cost of the deletion operation is \(c_d\).

3. Substitute: Substitutes (replaces) the \(i^{th}\) character in \(X\) with another character. For example, if \(X = \langle B, G \rangle\), then \(\text{Substitute}(X, 0, 'F')\) yields the string \(X' = \langle F, G \rangle\). The cost of the substitute operation is \(c_s\).

Provide a dynamic programming algorithm that finds the minimum-cost edit distance between two strings \(X[1..m]\) and \(Y[1..n]\). That is, find the minimum edit distance that transforms \(X\) to \(Y\). For example, if \(X = \text{‘Sunday’}\) and \(Y = \text{‘Saturday’}\), the minimum edit distance is 3 assuming that \(c_i = c_d = c_s = 1\); the three edit operations are show below.

1. \(X' = \text{substitute}(X, 2, 'r') = \text{‘Sunday’}\).

2. \(X'' = \text{insert}(X', 1, 't') = \text{‘Sturday’}\).

3. \(X''' = \text{insert}(X'', 1, 'a') = \text{‘Saturday’}=Y\).

For simplicity, your algorithm should only compute the cost of the minimum edit distance; i.e., the algorithm does not have to produce the sequence of edit operations.
1. Develop a recursive formula $D(i,j)$ that defines the cost of the minimum edit distance between two prefixes of the two strings, $X[1..i]$ and $Y[1..j]$. Explain how this formula produces an optimal answer.

2. Write a pseudo-code for a bottom-up dynamic programming algorithm based on the recursive formula.

3. Establish the running time of your bottom-up algorithm.

4. Apply your algorithm to the two inputs $X = \text{"Sunday"}$ and $Y = \text{"Saturday"}$, assuming $c_i = 0.7$, $c_d = 0.9$, $c_s = 1.0$. Use the following table to show all the intermediate values of $D(i,j)$ based on your formula. Highlight the final answer in the table.

Question 3 In the single-source all-destination shortest path algorithm, we described two algorithms, namely, Bellman-Ford and Dijkstra. We showed that Bellman-Ford has a higher asymptotic running time but can work with both positive and negative weights on edges. Dijkstra’s algorithm is more efficient but might fail if the graph contained edges with negative weights. A professor suggests the following approach to use Dijkstra’s algorithm with a graph containing negative weights.

1. Do one pass over all the edges to compute the minimum weight, say $w_{\min}$. 

2. If $w_{\text{min}} < 0$, make another pass over all the edges and update the weights by adding the value $-w_{\text{min}}$. This makes all the weights $\geq 0$.

3. Apply Dijkstra’s algorithm on the same graph with the updated weights.

4. Compute the correct weights of the computed shortest paths by subtracting $-w_{\text{min}}$ from the weight of each edge.

The reasoning here is that if we add a constant value to all edges, the shortest path will remain shortest with probably a different weight. Is this approach correct? That is, does it always produce the minimum-weight paths? If your answer is yes, argue that the algorithm always yields the correct answer. If your answer is no, provide a counter example that shows that this technique is wrong.