Homework 9: Dynamic Programming

Elias Gabriel

1. (9 points) In this question, you will develop (but not implement) a dynamic programming approach to solve the **minimum edit distance problem**. In the minimum edit distance problem, you are given two strings $s1$ and $s2$ and the goal is to find the minimum number of edits needed to transform $s1$ into $s2$, where a single edit consists of either (a) an insertion of a single character, (b) a deletion of a single character, or (c) a substitution of a single character.

   For example, if $s1$ = 'cake' and $s2$ = 'cat' then the minimum edit distance would be two edits: deleting 'e' and substituting 't' for 'k' (or equivalently deleting 'k' and substituting 't' for 'e').

   (a) (7 points) Give a set of dynamic programming equations to find the minimum edit distance between two strings. Be sure to state what your value function calculates. Then, in 3-5 sentences, argue correctness of your DP solution using the principle of optimality.

Solution: You can find the minimum edit distance between two strings using the following set of value equations, where each one calculates the number of changes necessary to assimilate the substring up until the given index.

$$D[i,-1] = i+1 \tag{1}$$

$$D[-1,j] = j+1 \tag{2}$$

$$D[0,0] = \begin{cases} 0 & s_{1_0} = s_{2_0} \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

$$D[i,j] = \begin{cases} D[i-1,j-1] & s_{1_i} = s_{2_j} \\ min(D[i-1,j-1], D[i-1,j], D[i,j-1]) + 1 & \text{otherwise} \end{cases} \tag{4}$$

Equations 1-3 are base cases, where 1 and 2 calculate the number of changes when one of the two strings is empty. In Equation 4, the default case specifies to take the minimum number of edits from the three possible actions following a mismatch: substitution, deletion, and insertion.

In the first two base cases, we know that the algorithm chooses the minimum number of changes needed since one of the substrings is empty. Likewise, we know that the third base case will compare two characters and yield 1 if they are different since you can at max only make a single change to a single character. In the general case, we can let the optimal number of changes at iteration o be Δ^*. If the two compared characters are equal, then Δ^* is also the optimal solution for iteration $o+1$. If the compared characters are not equal, then excluding the character and its cost of k we know that $\Delta^* - k$ is the optimal solution for iteration $o+1$. Therefore since the algorithm is constructed to select k to be the minimum number of edits to assimilate a character at each step, we know that the total combined steps must be the global minimum.

(b) (2 points) What is the runtime of calculating your DP solution?

Solution: Using a bottom-up approach, you would need to fill the entire lookup table with the cases for all $i \in s_1$ and $j \in s_2$, resulting in a total runtime of $O(M*N)$ where M and N are the lengths of the given strings.

2. (15 points) In this question, you will see and implement an example of dynamic programming for a problem that does *not* involve optimization. In the **wildcard matching problem**, you are given a pattern string $s1$ and a wildcard string $s2$. While $s1$ is a fixed string of $a-z$ characters, $s2$ may contain one or more wildcard characters $*$ which represent any possible substring (including the empty string). The goal of this problem is to find whether there exists a substitution into the wildcard characters such that the end result yields $s1$.

   For example, if $s1$ = 'lemondrop' and $s2$ = 'l*dr*p*' the answer would be True since we can substitute in 'emon', 'o', and " for each of the wildcard characters, respectively. However, if $s2$ = 'lem*m*dr*p' the answer would be False because there is no possible wildcard match that would generate $s1$.

   (a) (7 points) Give a set of dynamic programming equations to find whether or not there is a wildcard match between two strings $s1$ and $s2$. Be sure to state what your value function represents. Then, in 3-5 sentences, argue correctness of your DP solution using the principle of optimality. **Hint:** Your value function should evaluate to True or False.

Solution: You can determine if a given string s matches a given pattern p with the following set of equations, where each equation specifies whether or not a two characters are equivalent:

$$B[0,0] = \texttt{True} \tag{5}$$

$$B[i,0] = \texttt{False} \tag{6}$$

$$B[0,j] = \begin{cases} B[0,j-1] & p_j = * \\ \texttt{False} & \text{otherwise} \end{cases} \tag{7}$$

$$B[i,j] = \begin{cases} B[i-1,j-1] & s_i = p_j \\ B[i,j-1] \vee B[i-1,j] & p_j = * \\ \texttt{False} & \text{otherwise} \end{cases} \tag{8}$$

Equations 1-3 cover the three base cases: when the strings are empty, when just the pattern is empty, and when just the string is empty. If the pattern and string are blank, we know they are equivalent. If there is no pattern but a string, we know a match cannot exist. Finally, if there is no string but a pattern, we know a match exists if and only if the pattern contains only wildcards. As we iterate through the strings, there are three possibilities and four possible actions. If the current character matches the pattern (`True`), we can simply skip it since the final result R would go unchanged ($\texttt{True} \wedge \texttt{False} = \texttt{False}$). If the current character does not match and is not a wildcard, we know that the pattern up until the current index j cannot match the string up until the index i at iteration (i, j). In the case of a wildcard, it may represent either an existent character or an empty one. If R evaluated at iteration $(i, j-1)$ holds `True`, we know that the current wildcard must code for an empty character, that it is non-essential ($\texttt{True} \wedge \texttt{True} = \texttt{True}$), and that R is also the solution for (i, j). If R evaluated at iteration $(i-1, j)$ holds `True`, then we know that the current character of the string at i is non-essential and therefore the overall solution R does not change. By "removing" non-essential characters from the string and pattern, and because the algorithm always selects truthiness over falseness (a consequence of the logical or), we know that the final result strictly compares essential characters and therefore returns the correct answer.

(b) (2 points) What is the runtime of calculating your DP solution?

Solution: Using a bottom-up approach, the runtime for this algorithm is identical to the runtime of the first one. By filling the entire lookup table with the cases for all $i \in s$ and $j \in p$, the runtime is bounded by $O(M * N)$ where M and N are the lengths of the given string and pattern respectively.

(c) (6 points) Implement and test your dynamic program. For your test function, construct 5-7 test cases that you think contain different possible structures.

This assignment requires a **course assistant check-off** so be prepared to explain your code structure and to talk through how you constructed your tests.