CSE525 Monsoon 2020 — Ashima Garg (PhD19003)
Homework 7 Problem 1 — Pramil Panjawani (PhD19008)

You CANNOT consult any other person or online resource for solving the homework problems. You can definitely ask the instructor or TAs for hints and you are encourage to do so (in fact, you will get useful hints if you ask for help at least 1-2 days before the due date). If we find you guilty of academic dishonesty, penalty will be imposed as per institute guidelines.

(a) Describe a recursive backtracking algorithm to find the smallest number of symbols that can be removed from Y so that X is no longer a subsequence. Equivalently, your algorithm should find the longest subsequence of Y that is not a supersequence of X. For example, after removing two symbols from the string PENPINEAPPLEAPPLEPEN, the string PPAP is no longer a subsequence.

Solution: Explanation: Let X and Y be two strings of length k and n respectively. *Subsequence* calls *SubsequenceExist* to test whether X is subsequence of Y or not. *SubsequenceExist* first checks if one of them is zero then, return X is not subsequence of Y. Else, if the characters dont match in both the strings, then simply search remaining string X in exclude the character of Y and search in the remaining of Y. Else, if the characters match, then we have two possibilities whether to consider the current matched character in Y or not. *take* or *skip* represents the same.

```
⟨⟨Let X[1...k], Y[1...n] be global arrays where k ≤ n.⟩⟩
⟨⟨Let i = 0, indexes array X and j = 0 indexes array Y⟩⟩
def SubsequenceExist(X, Y, i, j, k, n):
  if k == 0 or n == 0:
      return False
  if X[i] ≠ Y[j]:⟨⟨If the current characters dont match, then⟩⟩
⟨⟨Exclude this character of Y, and search X in the remaining sequence of Y.⟩⟩
      return SubsequenceExist(X, Y, i, j + 1, k, n)
  if X[i] == Y[j]: ⟨⟨If the characters of both the strings match, then, ⟩⟩
⟨⟨take (consider) this character of Y, and match remaining sequence of X, in remaining sequence of Y⟩⟩
      take =SubsequenceExist(X, Y, i + 1, j + 1, k, n)
⟨⟨skip (skip) this character of Y, and match the sequence of X, in remaining sequence of Y⟩⟩
      skip =SubsequenceExist(X, Y, i, j + 1, k, n)
      return take or skip

def Subsequence(X, Y):
  return SubseqeunceExist(X, Y, i = 1, j = 1, len(X), len(Y))
```

Time Complexity:

$$
\begin{aligned}
&T(n) \leq 2T(n-1) + O(1)\\
&T(n) \leq 2(2T(n-2) + O(1)) + O(1)\\
&T(n) \leq 4T(n-2) + 2c + c\\
&T(n) \leq 2^k T(n-k) + c(2^{k-1} + 2^{k-2} + \ldots + 2^1 + 1)\\
&\qquad => Since number of levels will be n, therefore, n-k = 0, n = k, T(0) = 1\\
&T(n) \leq 2^n T(0) + 2^{n-1} + 2^{n-2} + \ldots + 2^1 + 1\\
&T(n) \leq 2^{n+1} - 1\\
&T(n) \leq O(2^{n+1})\\
&T(n) = O(2^n)
\end{aligned}
$$

■

CSE525 Monsoon 2020 — Ashima Garg (PhD19003)
Homework 7 Problem 2 — Pramil Panjawani (PhD19008)

(b) Describe a recursive backtracking algorithm to find the smallest number of symbols that can be removed from Y so that X is no longer a subsequence. Equivalently, your algorithm should find the longest subsequence of Y that is not a supersequence of X. For example, after removing two symbols from the string PENPINEAPPLEAPPLEPEN, the string PPAP is no longer a subsequence.

Solution: The solution presented is naive, as to get Y that is not supersequence of X, we check the presence of each character makes the resultant sequence the solution or not and finally return the longest such Y. It could have been optimized if we check the presence or absence of characters that are present in X. But that would just optimize the solution, since we are asked to give naive solution, the below solution should work.
Call $main$ function. If one of the two sequences is empty, then simply return 0. Else, check if X is supersequence of given Y or not, if not, then the length of given Y is the maximum. Else, call *longestNotSupersequence(*$X, Y, j, max\_len\_so\_far$*)* where, $max\_len\_so\_far$ stores the maximum length of Y which is not supersequence of X (initialized to 0). Base condition will be when we found a Y which is not supersequence of X, then update $max\_len\_so\_far$. Else, consider every character of Y to be included or not in the final sequence that does not form the supersequence of X.

```
def longestNotSupersequence(X, Y, j, max_len_so_far, result):
    if not Subsequence(X, Y): ⟨⟨If X is not subsequence of Y, then length of current Y is one of all posibilities, ⟩⟩
⟨⟨So, update max_len_so_far variable with maximum length found so far and length of current Y⟩⟩
        if max_len_so_far < len(Y):
            max_len_so_far = len(Y)
            result = Y
        return
⟨⟨If the current character is not present in Y, check the remaining sequence Y is supersequence of X or not.⟩⟩
    longestNotSupersequence(X, Y − Y[j], j − 1, max_len_so_far)
⟨⟨If the current character is included in Y, check the remaining sequence Y is supersequence of X or not.⟩⟩
    longestNotSupersequence(X, Y, j − 1, max_len_so_far)

def main(X, Y): ⟨⟨Main function that starts the execution⟩⟩
    if len(Y) == 0 or len(X) == 0: return 0 ⟨⟨If one of the sequence X or Y is empty, then return 0⟩⟩
    if not Subsequence(X, Y): return Y ⟨⟨If X is not subsequence of Y, then given length of given Y is the maximum⟩⟩
    max_len_so_far = 0, result = ""
    longestNotSupersequence(X, Y, len(Y), max_len_so_far, result)
    return result
```

From the previous part(HW7, Problem 1), the upper bound of the step *Subsequence(*X, Y*)* function is $O(2^n)$
Time Complexity:

$$
\begin{aligned}
T(n) &\leq 2T(n-1) + O(2^n) \\
T(n-1) &\leq 2T(n-2) + O(2^{n-1}) \\
T(n-2) &\leq 2T(n-3) + O(2^{n-2}) \\
T(n) &\leq 8T(n-3) + c(2^n + 2^{n-1} + 2^{n-2}) \\
T(n) &\leq 2^k T(n-k) + c(2^n + 2^{n-1} + 2^{n-2} + \ldots + 2^{n-(k-1)}) \\
&\quad Since no of levels will be n, therefore, n-k = 0, n = k, T(0) = 1 \\
T(n) &\leq 2^n T(0) + (2^n + 2^{n-1} + 2^{n-2} + \ldots + 2 + 1) \\
T(n) &\leq 2^n + (2^{n+1} - 1) \\
T(n) &\leq O(3 * (2^n) - 1) \\
T(n) &\leq O(2^n)
\end{aligned}
$$

■