CS 344 - Sections 1,2,3 - Fall 2018
Homework 5
Due Tuesday, Dec 4, 11:55pm
45 points total plus extra credit

VERY IMPORTANT: For all the dynamic programming problems below, be explicit about the following:

1. What the values in your table correspond to. So e.g. for longest increasing subsequence (from class) I would write something like: $T[i]$ is the length of the longest increasing sequence ending at $A[i]$

2. The DP-relation that computes one table entry in terms of other table entires. So for longest increasing subsequence I would write something like: $T[i] = \max_j T[j] + 1$, where the maximum is taken over all $j < i$ with $A[j] < A[i]$.

3. the order in which you compute the entries of the table

4. Which value you return in the end

5. for some of the problems you will be asked to also include pseudocode. For these problems, write out steps 1 and 2 before the pseudocode, as this will make your code much easier to follow.

1 Problem 1: Different Bills (10 points)

This problem is the same as the one from class, but with different bill numbers. Say that you had bills of value \$1, \$6, \$27, \$38, \$50. You want to solve the following problem MinBills(N)

- Input: some number N
- Output: the smallest number of bills you need to make \$N, *as well as the actual sequence of bills that you use*. If there are multiple possible sequences of bills, you only have to output one of them.

Write pseudocode for the above problem with run-time O(N). For this problem you have to write pseudocode, and you have to return the actual set of bills used.

NOTE: for this problem, to make your life simpler, you can assume that a negative index at an array always returns ∞. So e.g. $T[-3] = \infty$. (In real code you would have to be more careful (e.g. pad the array with a bunch of ∞ in the beginning), but for this problem I don't need you to think about these sorts of fringe cases.)

2 Problem 2: Knapsack (10 points)

You have a set of objects $S = \{s_1, s_2, ...s_n\}$. Each object s_i has a weight w_i and a value v_i. You have a knapsack that can carry a total weight of at most W. A valid knapsack is a set of objects $T \subseteq S$ for which $\sum_{s_i \in T} w_i \leq W$. The value of a knapsack T is then $\sum_{s_i \in T} v_i$. Your goal is to find the maximum possible value of a valid knapsack.

For example, say that your max allowed weight is W = 17, and you have 6 objects:

- Object s_1 has $w_1 = 14$ and $v_1 = 20$
- Object s_2 has $w_2 = 15$ and $v_2 = 14$
- Object s_3 has $w_3 = 7$ and $v_3 = 12$
- Object s_4 has $w_4 = 5$ and $v_4 = 7$
- Object s_5 has $w_5 = 4$ and $v_5 = 5$
- Object s_6 has $w_6 = 2$ and $v_6 = 2$

Then, the maximum value you can get is $12 + 7 + 5 = 24$: you get this by picking objects s_3, s_4, and s_5. note that the total weight of this knapsack is $7 + 5 + 4 = 16$, which is less than the maximum allowed 17, so it is indeed a valid knapsack.

the Question: Describe a dynamic programming algorithm for this problem that runs in time $O(nW)$. No pseudocode necessary for this one, and you don't have to return the items in the knapsack, just the maximum value. Make sure your description includes all the details described on the first page of this homework assignment

HINT: you will want to create a 2-dimensional table T of size n by W, and you will want T[i][j] to store the maximum value you can get if the available objects are $s_1, ..., s_i$, and the maxium weight of your knapsack is j.

Now, as in all dynamic programming, you will want to show how to compute T[i][j] in terms of previous values in the table. My suggestion for how to go about this: consider two options for T[i][j]: you can either include s_i in your knapsack, or you can not include s_i in the knapsack. Use previous values in the table T to figure out the value you would get in each of the two cases, and then take the best of the two cases.

3 Problem 3: Maximum Interval Sum (10 points)

given an array A of length n, and two indexes i, j with $i \leq j$, define Sum(i,j) $= A[i] + A[i+1] ... + A[j]$; if $i = j$ then define Sum(i,j) $= A[i]$.

Consider the following problem:

- Input: array A of length n (starting at $A[0]$]), where A can have positive and negative numbers.
- Output: the maximum possible value Sum(i,j)

So for example, if A = 3,-5,4,-2,-1,4,-3,2, then the output is 5, which can attained by the interval [4,-2,-1,4].

Describe a dynamic programming solution to this problem that runs in $O(n)$ time. Again no pseudocode necessary, and no need to return i and j; just returning the maximum sum is enough. Make sure your description includes all the details described on the first page of this homework assignment

4 Problem 4 (15 points):

You are given two strings of english letters S and T. Your goal is to find the longest *consecutive* sequence of letters that appears in both both S and T. For example, if S = temporarily and T = emporium, then the output is empor.

Describe a dynamic programming algorithm for this problem that runs in time O(length(S) $\cdot$ length(T)). For this problem, write pseudocode, and make sure to return the actual substring, not just its length. (If there are multiple longest substrings, you only have to return one of them.)

HINT 1: This problem is easier if you set $T[i, j]$ be the length of the longest common substring of $X_1, ..., X_i$ and $Y_1, ...Y_j$ that ends in X_i and Y_j. So in particular, if $X_i \neq Y_j$, then $T[i, j] = 0$, because a common substring cannot end in two different letters.

HINT 2: for this problem, returning the actual sequence doesn't require storing an additional table S; there is an easier method for this problem. That being said, you are welcome to use a separate S table if that's more comfortable for you.

5 Extra Credit

You are given an array $P = P[0]...P[n-1]$ of stock prices on n consecutive days. Your goal is to determine when it would have been best to buy and sell. Formally, you want to find indices i and j with $j > i$ that maximize $P[j] - P[i]$. If there are multiple pairs that achieve the maximum, you only have to output one of them.

Write pseudocode that solves this problem in time $O(n)$.