CS 344 - Sections 5,6,7,8 - Spring 2021
Homework 4
Due Monday, March 29, 4:00pm
120 points total + 5 points EC

Very Important: what to write for DP problems

For all the dynamic programming problems in this class, your solution must contain ALL of the following, in the exact order below.

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. How do you initialize the table: so for longest increasing subsequence I would write $T[0] = 1$.

4. Which entry of the table you return at the end of the algorithm. So for example for longest increasing subsequence I would write: return FindMax(T)

5. Pseudocode for the final algorithm.

1 Problem 1 (10 points)

For this problem (and this problem only), we assume that we are dealing with very large numbers, so arithmetic operations depend on the number of digits in the numbers. You can use the following primitives:

- Given any number x any single digit number d, you can compute $x+d$, $x-d$, xd and x/d in O(len(x)) time, where len(x) is the number of digits in x

- Given any number x you can computer $x \cdot 10^k$ in $O(k)$ time.
- Given any numbers x and y, you can compute $x + y$ and $x - y$ in time O(len(x) + len(y)).

Now, say that your library contains a function Square(x), which given a number x with n digits computes the number x^2 in time $O(n \log(n))$. Write pseudocode for a function Mult(x,y) that given two n digit numbers x and y computes $x \cdot y$ in time $O(n \log(n))$. You will want to make use of the Square(x) function in your pseudocode.

HINT: the final answer is just a few lines of pseudocode.

2 Problem 2: Different Bills (15 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 minimum 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 a dynamic program for this problem with running time $O(N)$. Make sure to include all the dynamic programming steps detailed at the top of the HW. For this problem 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.)

3 Problem 3 (25 points total)

Sally lives at the bottom of long stairs with n steps. Sally is quite nimble, so she can go up the stairs one step at a time, but she can also skip steps. Sally

goes up these stairs every day so she thinks about them a lot. She wonders *how many* different ways she has of going up the stairs.

Formally, Sally goes up the stairs in a sequence of moves. In a single move she can go up one step or two steps or three steps, all the way up to k steps, for some variable $k \leq n$. Write a dynamic program NumCombos(n,k) that determines how many possible sequences of moves there are for Sally to get to step n. She starts at step 0.

For example, if $n = 5$ and $k = 2$ then in each move Sally can go either one step or two steps. In this case, NumCombos(n,k) = 8 because there are seven ways: 11111,1112,1121,1211,122,2111,212,221.

Note that NumCombos(n,k) only outputs the *number of ways*: you do NOT have to enumerate all possible ways.

Part 1 (15 points) Write a dynamic program to solve NumCombos(n,k) in $O(nk)$ time. Make sure to include all elements of a dynamic program detailed at the top of the HW.

EXTRA CREDIT: if your algorithm runs in $O(n)$ time, you will receive 5 extra points. So if you give a correct answer with $O(nk)$ running time you will get 15/15 points. If you give a correct answer with $O(n)$ runtime, you will get 20/15 points. *You may only give one answer.*

Part 2 (10 points) Now say that Sally wants to make ≤ 100 moves total. Write a dynamic program NumCombosH(n,k) that determines the total possible number of ways Sally can climb the stairs with this additional restriction. The running time should be $O(nk)$. Again make sure to include all elements of a dynamic program detailed above.

Note that in this case, you might actually have NumCombosH(n,k) = 0. For example, if $n = 1000$ and $k = 9$ then there is just no way Sally is getting to the top in 100 moves.

For this problem, there is no extra credit for a faster algorithm.

NOTE: even though the numbers can get very large for this problem, you can assume all arithmetic operations take $O(1)$ time.

4 Problem 4: space saving – 15 points

For the most part, we haven't been optimizing for space in our dynamic programs. But for many of the problems we've discussed it is possible to get

away with much less space. The goal of this problem is for you do out one such example.

Recall the subsetsum problem from class.

- INPUT: set $S = \{s_1, ..., s_n\}$ of non-negative integers and a target number B.
- OUTPUT: set $S' \subseteq S$ with $\sum_{x \in S'} = B$ or FALSE if no such S' exists.

The algorithm we showed in class used $O(nB)$ time and $O(nB)$ space.

The Problem Write pseudocode for an algorithm that solves subset sum in $O(nB)$ time but only need $O(B)$ space in addition to the input itself. Your algorithm must return the actual set S'. *For this problem you ONLY need to write pseudocode – no need to do the other dynamic programming steps outlined above.*

5 Problem 5: Suitcase packing (25 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 suitcase that can carry a total weight of at most W. A valid suitcase is a set of objects $T \subseteq S$ for which $\sum_{s_i \in T} w_i \leq W$. The value of a suitcase T is then $\sum_{s_i \in T} v_i$. Your goal is to find the maximum possible value of a valid suitcase.

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 suitcase is $7 + 5 + 4 = 16$, which is less than the maximum allowed 17, so it is indeed a valid suitcase.

the Question: Describe a dynamic programming algorithm for this problem that runs in time $O(nW)$. As always, do the DP steps outlined at the top of the HW. *For this problem, you do not need to return the final items in the suitcase; you only need to return the total final value of the suitcase.*

HINT: the table you use is somewhat similar to Subset Sum from class, but instead of having each $T[i, j]$ store true/false you will want to it store an actual number. That number should tell you the "best" you can do for this suproblem.

The logic you use to compute $T[i, j]$ from previous $T[i', j']$ is also somewhat similar to subset sum, though more complicated.

6 Problem 6 (30 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. Note that empori is not a valid output because in string S the i does not appear right after empor.

Describe a dynamic programming algorithm for this problem that runs in time O(length(S) $\cdot$ length(T)). Make sure to include all four elements of a dynamic program detailed above. *For this problem, you pseudocode should return the actual substring, not just its length.* (If there are multiple longest substrings, you only have to return one of them.)

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