CS/ECE 374 A (Spring 2020)
Homework 6 (due Mar 12 Thursday at 10am)

Instructions: As in previous homeworks.

General Note: In any dynamic programming solution, you should follow the steps below (if we explicitly state that pseudocode is not required, then step 4 may be skipped):

1. first give a clear, precise definition of the subproblems (i.e., what the recursive function is intended to compute);
2. then derive a recursive formula to solve the subproblems (including base cases), with justification or proof of correctness if the formula is not obvious;
3. specify a valid evaluation order;
4. give pseudocode to evaluate your recursive formula bottom-up (with loops instead of recursion); and
5. analyze the running time.

Do not jump to pseudocode immediately. Never skip step 1!

Problem 6.1:

(a) (7.5 points) Given an array A of n numbers, describe an efficient dynamic programming algorithm to find the length of a longest subsequence S that is (strictly) increasing, such that both S and S^R are subsequences of A. Here, S^R denotes the reverse of S.

(For example, for $A = \langle 1, 2, 3, 4 \rangle$, the optimal length is 1; for $A = \langle 1, 15, 8, 2, 3, 5, 4, 3, 8, 10, 1, 9 \rangle$, the optimal length is 4, because both $\langle 1, 3, 5, 8 \rangle$ and $\langle 8, 5, 3, 1 \rangle$ are subsequences.)

Hint: for the definition of subproblems, let $L(i, j, k)$ be the length of a longest increasing subsequence such that S is a subsequence of $A[i, \ldots, n]$ and S^R is a subsequence of $A[1, \ldots, j]$, and all elements in S are greater than $A[k]$.

(b) (2.5 points) Give pseudocode to output an optimal subsequence S (not just its length).

Problem 6.2: We have n jobs, where job i starts at time t_i and has value $v_i > 0$, with $t_1 < t_2 < \cdots < t_n$. Each job requires Δ units of time to complete (and so job i ends at time $t_i + \Delta$). We have 2 servers. The goal is to choose a subset of jobs with the largest total value that can be handled by the 2 servers.

It is not difficult to see that an optimal strategy would have the servers alternately handle the chosen jobs—in other words, if the selected jobs are $i_1, \ldots, i_k$ with $i_1 < i_2 < \cdots < i_k$, server 1 would handle jobs $i_1, i_3, \ldots$ and server 2 would handle jobs $i_2, i_4, \ldots$ (You don't need to prove this fact.)

The problem can thus be reformulated as follows: choose $i_1 < i_2 < \cdots < i_k$ to maximize $v_{i_1} + v_{i_2} + \cdots + v_{i_k}$, subject to the constraint that $t_{i_3} - t_{i_1}, t_{i_4} - t_{i_2}, \ldots, t_{i_k} - t_{i_{k-2}}$ are all greater than Δ.

Describe an efficient dynamic programming algorithm to compute the value of an optimal solution. (You do not need to output the jobs in an optimal subset. $O(n^3)$ time will be good enough to get full credit.)

Problem 6.3: We are given an $n \times n$ grid. At each position (i, j) (with $1 \le i, j \le n$), there is a prize money of w_{ij} dollars (with $w_{ij} \ge 0$). Your car is initially at the lower-left corner (position $(1, 1)$) and *can only go up or right*. Your goal is to travel to the upper-right corner (position (n, n)) and collect as much money as possible.

(a) (5.0 points) Describe an efficient dynamic programming algorithm to compute the value of an optimal solution. Define subproblems, derive recursive formulas, specify evaluation order, etc. as usual, but pseudocode is not required for this question. (You do not need to output an optimal path.)

(b) (5.0 points) Describe an efficient algorithm to compute the optimal value for a variant of the problem, where we are given a number k, and there is an additional constraint that the path can make at most k turns. Again, pseudocode is not required. (You may assume that $k \le 2n$, but k may *not* be a constant.)

(In the following example, the optimal value for (a) is $0 + 4 + 0 + 5 + 2 + 11 + 7 + 8 + 0 = 37$; this solution, shown in blue, uses 5 turns. For (b) with $k = 2$, the optimal value with 2 turns is $0 + 4 + 0 + 5 + 2 + 9 + 15 + 0 + 0 = 35$.)

8	0	15	0	0
2	4	9	7	8
3	1	2	11	0
0	0	5	0	0
0	4	0	0	16