CS/ECE 374 A (Spring 2022)
Homework 6 (due March 10 Thursday at 10am)

Instructions: As in previous homeworks.

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.

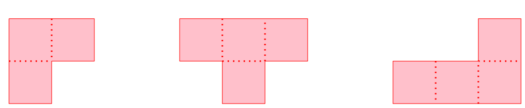
Do not jump to pseudocode immediately. Never skip step 1!

Problem 6.1: For a sequence $\langle b_1, \ldots, b_m \rangle$, an *alternation* is an index $i \in \{2, \ldots, m-1\}$ such that ($b_{i-1} < b_i$ and $b_i > b_{i+1}$) or ($b_{i-1} > b_i$ and $b_i < b_{i+1}$).

(a) (80 pts) Given a sequence $\langle a_1, \ldots, a_n \rangle$ and an integer $k \leq n-1$, we want to compute a longest subsequence that has at most k alternations.
(For example, for the input sequence $\langle 3, 1, 6, 8, 2, 10, 9, 4, 5, 12, 7, 11 \rangle$ and $k = 2$, an optimal subsequence is $\langle 1, 6, 8, 10, 9, 4, 5, 7, 11 \rangle$, which has 2 alternations.)
Describe an $O(kn^2)$-time dynamic programming algorithm to solve this problem.[1] In this part, your algorithm only needs to output the optimal value (i.e., the length of the longest subsequence).

(b) (20 pts) Give pseudocode to also output an optimal subsequence.

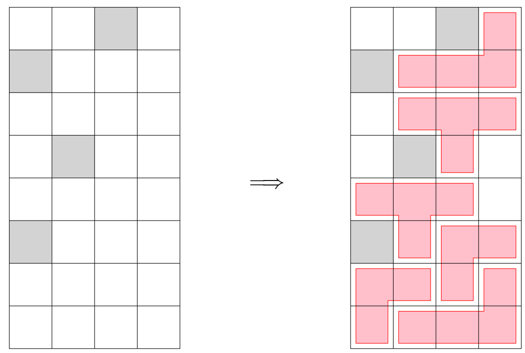
Problem 6.2: We have an $n \times 4$ grid, with n rows and 4 columns. We are given an $n \times 4$ matrix F, where $F[i, j] = 1$ indicates that the grid cell at the i-th row and j-th column is *forbidden*, and $F[i, j] = 0$ indicates that the cell is "allowed". The goal is to cover the maximum number of grid cells using shapes of the following three types (we are *not* allowed to rotate these shapes):



The constraints are: (i) no forbidden cells are covered, and (ii) each cell is covered at most once (i.e., the shapes can't overlap).

[1] You may assume that all the a_i's are distinct.

In the following example with $n = 8$, the forbidden cells are shaded in gray, and the solution shown in red covers 22 cells, but is not optimal (can you do better?).



input

a (suboptimal) solution

(a) (90 pts) Design and analyze an efficient dynamic programming algorithm to solve this problem. Your algorithm only needs to output the optimal value.

Hint: define a subproblem for each $i = 1, \ldots, n$ and each of the 16 possible "states" that the current row may be in...

(b) (10 pts) If we change the problem to allow the shapes to be rotated (for example, the "T" shape can be rotated in 4 ways), how would you change the definition of your subproblems, and how many subproblems would you need as a function of n? (For this part, don't give the recursive formula or the actual algorithm, since the details are messier.)