CS/ECE 374 A (Spring 2022)
Homework 6 Solutions

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.

Solution:

(a) *Definition of subproblems.*

For each i, j with $1 \leq i < j \leq n+1$ and each $h \in \{0, \ldots, k\}$, let $L^+(i, j, h)$ be the length of a longest subsequence of $\langle a_i, a_j, a_{j+1}, \ldots, a_n \rangle$ such that the number of alternations is at most h and the first element in the subsequence is a_i and the second element (if exists) is greater than a_i.

For each i, j with $1 \leq i < j \leq n+1$ and each $h \in \{0, \ldots, k\}$, let $L^-(i, j, h)$ be the length of a longest subsequence of $\langle a_i, a_j, a_{j+1}, \ldots, a_n \rangle$ such that the number of alternations is at most h and the first element in the subsequence is a_i and the second element (if exists) is less than a_i.

The final answer we want is $\max_{i=1}^{n} \max\{L^+(i, i+1, k), L^-(i, i+1, k)\}$.

Base cases. $L^+(i, n+1, h) = L^-(i, n+1, h) = 1$ for each $i \in \{1, \ldots, n\}$ and $h \in \{0, \ldots, k\}$.

Recursive formula. For each i, j with $1 \leq i < j \leq n$ and $h \in \{0, \ldots, k\}$,

$$L^+(i,j,h) = \begin{cases} \max\{L^+(i, j+1, h),\ L^+(j, j+1, h) + 1,\ L^-(j, j+1, h-1) + 1\} & \text{if } a_j > a_i \text{ and } h \geq 1 \\ \max\{L^+(i, j+1, h),\ L^+(j, j+1, h) + 1\} & \text{if } a_j > a_i \text{ and } h = 0 \\ L^+(i, j+1, h) & \text{otherwise} \end{cases}$$

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

$$
L^-(i,j,h) \;=\; \begin{cases} \max\{L^-(i,j+1,h),\ L^-(j,j+1,h)+1,\ L^+(j,j+1,h-1)+1\} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{if } a_j < a_i \text{ and } h \geq 1 \\ \max\{L^-(i,j+1,h),\ L^-(j,j+1,h)+1\} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{if } a_j < a_i \text{ and } h = 0 \\ L^-(i,j+1,h) \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{otherwise} \end{cases}
$$

Justification. Consider the optimal solution corresponding to $L^+(i,j,h)$.

- Case 1: the second element in the optimal subsequence is not a_j. Then $L^+(i,j,h) = L^+(i,j+1,h)$.
- Case 2: the second element in the optimal subsequence is a_j and the third element (if exists) is greater than a_j. This case is applicable only when $a_j > a_i$. In this case, $L^+(i,j,h) = L^+(j,j+1,h)+1$.
- Case 3: the second element in the optimal subsequence is a_j and the third element (if exists) is less than a_j. This case is applicable only when $a_j > a_i$ and $h \geq 1$. In this case, $L^+(i,j,h) = L^-(j,j+1,h-1)+1$ (since the first three elements in the optimal subsequence form an alternation, so after excluding a_i, we are allowed at most $h-1$ alternations).

We don't know which case we are in beforehand. So, we take max over all applicable cases.

The justification for $L^-(i,j,h)$ is similar.

Evaluation order. In order of decreasing j.

Pseudocode.

1. for $i = 1$ to n do for $h = 0$ to k do $L^+[i,n+1,h] = L^-[i,n+1,h] = 1$
2. for $j = n$ down to 2 do
3.   for $i = 1$ to $j-1$ do
4.   for $h = 0$ to k do
5.   if $a_j > a_i$ and $h \geq 1$ then
   $L^+[i,j,h] = \max\{L^+[i,j+1,h],\ L^+[j,j+1,h]+1,\ L^-[j,j+1,h-1]+1\}$
6.   else if $a_j > a_i$ and $h = 0$ then
   $L^+[i,j,h] = \max\{L^+[i,j+1,h],\ L^+[j,j+1,h]+1\}$
7.   else $L^+[i,j,h] = L^+[i,j+1,h]$
8.   if $a_j < a_i$ and $h \geq 1$ then
   $L^-[i,j,h] = \max\{L^-[i,j+1,h],\ L^-[j,j+1,h]+1,\ L^+[j,j+1,h-1]+1\}$
9.   else if $a_j < a_i$ and $h = 0$ then
   $L^-[i,j,h] = \max\{L^-[i,j+1,h],\ L^-[j,j+1,h]+1\}$
10.   else $L^-[i,j,h] = L^-[i,j+1,h]$
11. $\ell^* = -\infty$, $i^* = 0$
12. for $i = 1$ to n do
13.   if $L^+[i,i+1,k] > \ell^*$ then $\ell^* = L^+[i,i+1,k]$, $i^* = i$, $sign^* = +$
14.   if $L^-[i,i+1,k] > \ell^*$ then $\ell^* = L^-[i,i+1,k]$, $i^* = i$, $sign^* = -$
15. return ℓ^*

Analysis. Line 1 takes $O(kn)$ time. Lines 2–10 take $O(kn^2)$ time. Lines 11–13 take $O(n)$ time. Total time: $O(kn^2)$.

(b) After running the algorithm in (a), we call $\textsc{OutputSubseq}^+(i^*, i^*+1, k)$ if $sign^* = +$, and $\textsc{OutputSubseq}^-(i^*, i^*+1, k)$ if $sign^* = -$:

$\textsc{OutputSubseq}^+(i, j, h)$:

1. if $j = n+1$ then output a_i and return
2. if $L^+[i, j, h] = L^+[i, j+1, h]$ then call $\textsc{OutputSubseq}^+(i, j+1, h)$
3. else if $L^+[i, j, h] = L^+[j, j+1, h] + 1$ then output a_i and call $\textsc{OutputSubseq}^+(j, j+1, h)$
4. else output a_i and call $\textsc{OutputSubseq}^-(j, j+1, h-1)$

$\textsc{OutputSubseq}^-(i, j, h)$:

1. if $j = n+1$ then output a_i and return
2. if $L^-[i, j, h] = L^-[i, j+1, h]$ then call $\textsc{OutputSubseq}^-(i, j+1, h)$
3. else if $L^-[i, j, h] = L^-[j, j+1, h] + 1$ then output a_i and call $\textsc{OutputSubseq}^-(j, j+1, h)$
4. else output a_i and call $\textsc{OutputSubseq}^+(j, j+1, h-1)$

(This takes $O(n)$ additional time.)

Remarks. The formulas could be rewritten more compactly using sentinels (e.g., setting $L^+[i, j, -1] = L^-[i, j, -1] = -\infty$) and symmetry (to avoid repetition in the handling of L^+ vs. L^-). We could also alternatively work "backward" instead of "forward" (working with prefixes instead of suffixes).

Alternate Solution (sketch): In the following alternate solution, the number of subproblems is smaller ($O(kn)$ instead of $O(kn^2)$), but the time needed per subproblem is increased ($O(n)$ instead of $O(1)$).

Definition of subproblems.

For each $i \in \{1, \ldots, n\}$ and $h \in \{0, \ldots, k\}$, let $L^+(i, h)$ be the length of a longest subsequence of $\langle a_i, a_{i+1}, \ldots, a_n \rangle$ such that the number of alternations is at most h and the first element in the subsequence is a_i and the second element (if exists) is greater than a_i.

For each $i \in \{1, \ldots, n\}$ and $h \in \{0, \ldots, k\}$, let $L^-(i, h)$ be the length of a longest subsequence of $\langle a_i, a_{i+1}, \ldots, a_n \rangle$ such that the number of alternations is at most h and the first element in the subsequence is a_i and the second element (if exists) is greater than a_i.

The final answer we want is $\max_{i=1}^n \max\{L^+(i, k), L^-(i, k)\}$.

Recursive formula. For each $i \in \{1, \ldots, n\}$ and $h \in \{0, \ldots, k\}$,

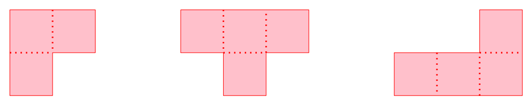
$$L^+(i, h) = \begin{cases} \max\{1,\ \max_{j>i:\ a_j > a_i} \max\{L^+(j, h) + 1,\ L^-(j, h-1) + 1\}\} & \text{if } h \geq 1 \\ \max\{1,\ \max_{j>i:\ a_j > a_i} (L^+(j, h) + 1)\} & \text{if } h = 0 \end{cases}$$

$$L^-(i, h) = \begin{cases} \max\{1,\ \max_{j>i:\ a_j < a_i} \max\{L^-(j, h) + 1,\ L^+(j, h-1) + 1\}\} & \text{if } h \geq 1 \\ \max\{1,\ \max_{j>i:\ a_j < a_i} (L^-(j, h) + 1)\} & \text{if } h = 0 \end{cases}$$

I'll omit justification, pseudocode, etc., but the running time of this alternate solution is still $O(kn^2)$.

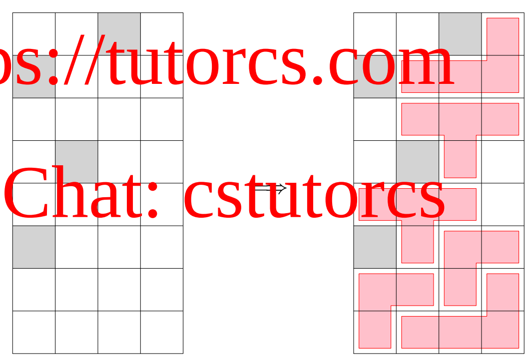
Remark. There are alternative solutions that are even faster, with running time $O(kn \log n)$...

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).

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 $i = 1, \dots, n$ 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.)

Solution:

(a) *Definition of subproblems.* For each $i \in \{1, \ldots, n\}$ and $a_1, a_2, a_3, a_4 \in \{0, 1\}$, let $M[i, a_1a_2a_3a_4]$ be the maximum number of grid cells that can be covered, subject to the constraints that the covered cells are all in the bottom i rows, and for each $j \in \{1, 2, 3, 4\}$ with $a_j = 1$, the cell at the i-th row and j-th column cannot be covered.

The final answer we want is $M[n, 0000]$.

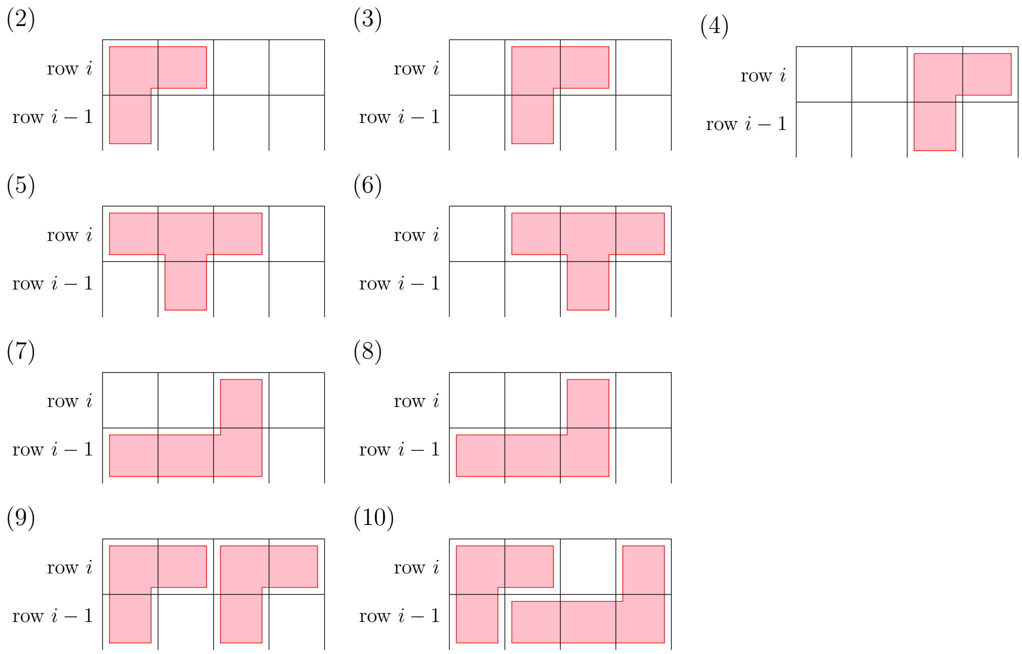
Base cases. $M[1, a_1a_2a_3a_4] = 0$ for each $a_1, a_2, a_3, a_4 \in \{0, 1\}$.

Recursive formula. For each $i \in \{2, \ldots, n\}$ and $a_1, a_2, a_3, a_4 \in \{0, 1\}$,

$$
\begin{aligned}
&M[i, a_1a_2a_3a_4] \\
&= \max \left\{ \begin{array}{lll}
M[i-1, 0000] & & (1) \\
M[i-1, 1000] + 3 & \text{if } a_1 = a_2 = F[i,1] = F[i,2] = F[i-1,1] = 0 & (2) \\
M[i-1, 0100] + 3 & \text{if } a_2 = a_3 = F[i,2] = F[i,3] = F[i-1,2] = 0 & (3) \\
M[i-1, 0010] + 3 & \text{if } a_3 = a_4 = F[i,3] = F[i,4] = F[i-1,3] = 0 & (4) \\
M[i-1, 0100] + 4 & \text{if } a_1 = a_2 = a_3 = F[i,1] = F[i,2] = F[i,3] & (5) \\
 & \quad = F[i-1,2] = 0 & \\
M[i-1, 0010] + 4 & \text{if } a_2 = a_3 = a_4 = F[i,2] = F[i,3] = F[i,4] & (6) \\
 & \quad = F[i-1,3] = 0 & \\
M[i-1, 1110] + 4 & \text{if } a_3 = F[i,3] = F[i-1,1] = F[i-1,2] & (7) \\
 & \quad = F[i-1,3] = 0 & \\
M[i-1, 0111] + 4 & \text{if } a_4 = F[i,4] = F[i-1,2] = F[i-1,3] & (8) \\
 & \quad = F[i-1,4] = 0 & \\
M[i-1, 1010] + 6 & \text{if } a_1 = a_2 = a_3 = a_4 = F[i,1] = F[i,2] & (9) \\
 & \quad = F[i,3] = F[i,4] = F[i-1,1] & \\
 & \quad = F[i-1,3] = 0 & \\
M[i-1, 1111] + 7 & \text{if } a_1 = a_2 = a_4 = F[i,1] = F[i,2] = F[i,4] & (10) \\
 & \quad = F[i-1,1] = F[i-1,2] = F[i-1,3] & \\
 & \quad = F[i-1,4] = 0 &
\end{array} \right.
\end{aligned}
$$

(In the above, we are taking the maximum of up to 10 terms, omitting the terms for which the corresponding conditions are not true.)

Justification. Consider the optimal solution corresponding to $M[i, a_1a_2a_3a_4]$. We divide into 10 cases, nine of which are depicted in the figure below:



- Case 1: the optimal solution does not cover any of the cells in the i-th row. Then $M[i, a_1a_2a_3a_4] = M[i-1, 0000]$.
- Case 2: the optimal solution covers the leftmost two cells of i-th row using the “Γ” shape but left the remaining two cells uncovered. This case is applicable only if $a_1 = a_2 = F[i,1] = F[i,2] = F[i-1,1] = 0$. In this case, $M[i, a_1a_2a_3a_4] = M[i-1, 1000] + 3$.
- Case 3 or 4: similar.
- Case 5: the optimal solution covers the leftmost three cells of i-th row using the “T” shape but left the fourth cell uncovered. This case is applicable only if $a_1 = a_2 = a_3 = F[i,1] = F[i,2] = F[i,3] = F[i-1,2] = 0$. In this case, $M[i, a_1a_2a_3a_4] = M[i-1, 0100] + 4$.
- Case 6: similar.
- Case 7: the optimal solution covers the third leftmost cell of i-th row using the “__|” shape but left the remaining three cells uncovered. This case is applicable only if $a_3 = F[i,3] = F[i-1,1] = F[i-1,2] = F[i-1,3] = 0$. In this case, $M[i, a_1a_2a_3a_4] = M[i-1, 1110] + 4$.
- Case 8: similar.
- Case 9: the optimal solution covers the entire i-th row using two “Γ” shapes. This case is applicable only if $a_1 = a_2 = a_3 = a_4 = F[i,1] = F[i,2] = F[i,3] = F[i,4] = F[i-1,1] = F[i-1,3] = 0$. In this case, $M[i, a_1a_2a_3a_4] = M[i-1, 1010] + 3 + 3$.
- Case 10: the optimal solution covers the three cells of the i-th row using one “Γ” shape and one “__|” shape. This case is applicable only if $a_1 = a_2 = a_4 = F[i,1] = F[i,2] = F[i,4] = F[i-1,1] = F[i-1,2] = F[i-1,3] = F[i-1,4] = 0$. In this case, $M[i, a_1a_2a_3a_4] = M[i-1, 1111] + 3 + 4$.

We don't know which case we are in beforehand. So, we take max over all applicable cases.

Evaluation order. In order of increasing i.

Pseudocode.

1. for each $a_1, a_2, a_3, a_4 \in \{0,1\}$ do $M[1, a_1a_2a_3a_4] = 0$
2. for $i = 2$ to n do
3.   for each $a_1, a_2, a_3, a_4 \in \{0,1\}$ do
4. $m = M[i-1, 0000]$
5.   if $a_1 = a_2 = F[i,1] = F[i,2] = F[i-1,1] = 0$ then
            $m = \max\{m, M[i-1, 1000] + 3\}$
6.   if $a_2 = a_3 = F[i,2] = F[i,3] = F[i-1,2] = 0$ then
            $m = \max\{m, M[i-1, 0100] + 3\}$
7.   if $a_3 = a_4 = F[i,3] = F[i,4] = F[i-1,3] = 0$ then
            $m = \max\{m, M[i-1, 0010] + 3\}$
8.   if $a_1 = a_2 = a_3 = F[i,1] = F[i,2] = F[i,3] = F[i-1,2] = 0$ then
            $m = \max\{m, M[i-1, 0100] + 4\}$
9.   if $a_2 = a_3 = a_4 = F[i,2] = F[i,3] = F[i,4] = F[i-1,3] = 0$ then
            $m = \max\{m, M[i-1, 0010] + 4\}$
10.   if $a_3 = F[i,3] = F[i-1,1] = F[i-1,2] = F[i-1,3] = 0$ then
            $m = \max\{m, M[i-1, 1110] + 4\}$
11.   if $a_4 = F[i,4] = F[i-1,2] = F[i-1,3] = F[i-1,4] = 0$ then
            $m = \max\{m, M[i-1, 0111] + 4\}$
12.   if $a_1 = a_2 = a_3 = a_4 = F[i,1] = F[i,2] = F[i,3] = F[i,4] = F[i-1,1]$
            $= F[i-1,3] = 0$ then
            $m = \max\{m, M[i-1, 1010] + 6\}$
13.   if $a_1 = a_2 = a_4 = F[i,1] = F[i,2] = F[i,4] = F[i-1,1]$
            $= F[i-1,2] = F[i-1,3] = F[i-1,4] = 0$ then
            $m = \max\{m, M[i-1, 1111] + 7\}$
14. $M[i, a_1a_2a_3a_4] = m$
15. return $M[n, 0000]$

Analysis. Line 1 takes $O(1)$ time. Lines 2–13 takes $O(n \cdot 16) = O(n)$ time. Total time: $O(n)$.

Remarks. The number of table entries can be slightly reduced by noting that only 9 of the 16 candidates for $a_1a_2a_3a_4$ may arise during recursion. We could also alternatively work "forward" instead of "backward" (covering rows i to n instead of rows 1 to i in the definition of subproblems).

(b) Intuitively, we need to "remember" not just the state of two rows instead of one, since a rotated shape may span three rows instead of two.

Formally, we change the definition of subproblems as follows: For each $i \in \{1, \ldots, n\}$ and $a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4 \in \{0,1\}$, let $M[i, a_1a_2a_3a_4b_1b_2b_3b_4]$ be the maximum number of grid cells that can be covered, subject to the constraints that the covered cells are all in the bottom i rows, and for each $j \in \{1,2,3,4\}$ with $a_j = 1$, the cell at the i-th row and j-th column cannot be covered, and for each $j \in \{1,2,3,4\}$ with $b_j = 1$, the cell at the $(i-1)$-th row and j-th column cannot be covered.

The number of subproblems is $2^8 n$, which is still $O(n)$.
(The number of cases increases, but in principle, we should still get an $O(n)$-time algorithm.)