HW6 - S16

Handed out 5-10 Chris Troiano 5-17, beg. of class

1. Determine the longest common subsequence of $x = (1, 1, 0, 0, 1, 1, 0, 1)$ and $y = (0, 1, 0, 1, 1, 0, 0, 1, 0)$. Build the table $C(i, j)$ of the dynamic programming algorithm for these two strings, where $C(i, j)$ denotes the length of the longest common subsequence between $x[1 \dots i]$ and $y[1 \dots j]$. Also produce the table of "arrows" that lets you recover the solution.
   Solution:

2. KT #20 p 329
   Solution:

$$opt(h, i) = \begin{cases} 0, & \text{if i = 0} \\ f_i(0), & \text{if h' = 0} \\ max(f_i(h') + opt(h - h', i - 1)), & \text{for } 0 \leq \text{h'} \leq \text{h} \end{cases} \quad (1)$$

If $i = 0$, there aren't any projects, and you get no grade.
If $h = 0$, you have 0 hours to spend studying, and you receive a grade $g = f_i(0)$.
Otherwise, you iterate from 0 to h on each project, taking the max grade with the number of hours spent on the current project, with respect to your previous project $i - 1$.
Subproblems: Given a total number of hours h, what is the best number of hours h' that we can spend on each project i, that will maximize our grade. This depends on the grade we

received on the previous project.

Time: The time bound is $O(h \cdot n \cdot g)$. This is because we must fill out $h \cdot n$ spaces on the table with the $f_i(h')$ value. Which I assume must take O(g). Therefore our algorithm takes $O(h \cdot n \cdot g)$ time.

3. A chess master plays n games at a chess competition, each against a different opponent. Based on their past performances, we can estimate the probabilities of each opponent beating the master: $p_1, \ldots, p_n \in [0, 1]$. Describe an algorithm, which given integer k returns the probability that the master wins exactly k out of n games. Then, use it to find the probability of the chess master winning the majority of the games (at least $\lceil \frac{n}{2} \rceil$).
Solution:

$$opt(i,k) = \begin{cases} (1 - P_i) \cdot opt(i-1, k-1) + (P_i) \cdot opt(i-1, k), & \text{for } i > 0 \text{ and } k \geq 0 \end{cases} \tag{2}$$

If the grand-master wins a game we decrement k and i, if he/she loses, then we just decrement i.

To find the probability of winning at least $\lceil \frac{n}{2} \rceil$, we fill out the table for $opt(n, n)$ and then add up k probabilities for $\lceil \frac{n}{2} \rceil \leq k \leq n$.

4. You're playing the game of tetris with two block shapes: one is the corner/L-shape block and one a small square block, as shown on the figure. The blocks are allowed to be rotated by $90°, 180°$, or $270°$ degrees You're supposed to completely fill up you're game area of width $n \geq 1$ up to the height 2, without any blocks sticking out. The Figure shows an example with $n = 10$. Design a dynamic programming algorithm, that for given n counts the number of possible ways to cover a $2 \times n$ rectangle with the two types of blocks.
Solution:

$$opt(n) = \begin{cases} 1, & \text{if n = 0} \\ 1, & \text{if n = 1} \\ opt(n-1) + 5 \cdot opt(n-2) + 2 \cdot opt(n-3) & \text{otherwise} \end{cases} \tag{3}$$

If n = 0, then there is only 1 way to fill out the array.
If n = 1, then there is only 1 way to fill out the array; with 2 cubes.

Otherwise, if there are 2 empty array spaces, there are 5 ways to fill it out; 4 ways with the L-shape block and a cube, and then 1 way with 4 cubes. If there are 3 empty array spaces, then there are 2 ways to fill it out, with 2 L-shaped blocks.
Depending on how many empty spaces we need to fill up, determines the number of permutations. The subproblems are related because they depend on the previous decision to determine which pieces are used.