Polytechnic Institute of NYU *CS6033: Design&Analysis of Algorithms I — Fall 2013*
Prof. Boris Aronov (aronov@poly.edu) RH315 Mon 6-8:30pm
Office: 2MTC 10.008 (718) 260-3092 Office hours: Mon 4-6pm

Homework #6: Dynamic programming and graphs, mostly.
Due November 18, 2013.
Refer to HOMEWORK RULES on PIAZZA.

#1: Dynamic programming example (chain matrix product): Given matrices A_1, A_2, A_3, A_4, A_5, A_6 of dimensions 2×10, 10×20, 20×50, 50×3, 3×100, 100×4, respectively, find the best order to do the product $A_1 \times A_2 \times A_3 \times A_4 \times A_5 \times A_6$. Show the two tables m and s constructed by the algorithm from the book and the final parenthesization.

#2: Dynamic programming example (longest common subsequence): Given the following input strings: `ALGORITHMS` and `MATHEMATICS`, compute a longest common subsequence using the algorithm from the book. Show the combined table (called c and b in the book) and a longest subsequence. Is there more than one longest subsequence?

#3: Consider the following variant of the rod-cutting problem:

> You are given a rod of length n inches. You have to decide how to cut it into pieces so that the total profit you make from selling pieces is maximized. You are given a table p_i, $i = 1, \ldots, n$, where the number p_i gives the price you can sell an i-inch piece for; assume $p_i > 0$.
>
> However, unlike the version in Section 15.1 in the book, you have to pay c, a given positive number, for each cut you make. So, for example, if you cut the rod into four pieces, of lengths $3, 4, 11, 5$, you made three cuts, so the profit you made is $p_3 + p_4 + p_{11} + p_5 - 3c$.
>
> The version described in class corresponds to setting $c = 0$.

The goal is to maximize profit. Let $P(i)$ be the maximum profit one can make from a rod of length i, for $i = 0, \ldots, n$. Give the recurrence defining P. Explain how to evaluate it efficiently. Give worst-case running time.

Reminder: Do not blindly repeat things from the book or the class. Show/emphasize what is different.

#4: Consider the following turn-based racing game:

> The race takes place on a track n units long, with $n + 1$ positions numbered 0 through n. You start at position 0 and move along the track, always forward. You finish the race when you reach position n ("finish position"), or go past it. You start with speed 0. You move one turn at a time. In a turn, if you are at position i and have speed s, you move to position $i + s$. You may also choose to speed up by one (go from speed s to $s + 1$), slow down (go from speed s to $s - 1$, if $s > 1$; you cannot slow down if your current speed is 0), or keep the current speed; this will be your speed for the *next turn*.
>
> The objective is finish the race as fast as possible (in as few turns as possible).

Unfortunately, there are some obstacles on the road. To model the obstacles, you are given two arrays MIN and MAX, with indices 0 through n. MIN$[i] = a$ means that when your car arrives at position i, or passes it in the current turn, you have to have speed at least a. Similarly MAX$[i] = b$ means that the speed cannot be larger than b when you arrive at or pass position i.

Design a dynamic-programming algorithm that finds the fewest number of turns that one needs to arrive at the finish line, given the number n and arrays MIN and MAX as input. Note that it may be impossible to finish the game, for some combinations of constraints.

Consider the following as a subproblem: find an optimal (i.e., one using minimum number of turns) way to, starting from 0, arrive a position i with speed s. Let turns(i, s) be the minimum number of turns required to reach position i arriving at speed s.

(i) Give a recursive definition of turns(i, s); do not forget the base case. *Hint*: What had to be the last move, in order for you to arrive at (i, s)?

It will be messy, due to the need to check min- and max-speed.

(ii) At position i, $i = 0, \ldots, n$, what is the maximum possible speed you can have, ignoring speed limits?

(iii) Armed with answers to (i) and (ii), how can you solve the dynamic programming problem? How fast is your algorithm?

(iv) An idea to speed up the algorithm: you need to repeatedly determine if, for a given pair of positions i and $j > i$, one can move at speed s and not violate any speed limits. This is the same as asking "Is it true that between i and j all the MIN values are at most s and all the MAX values are at least s?". If this is done brute-force, it would take time $\Theta(j - i + 1)$, which is linear in the worst case. I claim that (a very simple modification of) an augmented binary search structure we already discussed can solve the problem in logarithmic time, for a fixed triple (i, j, s). Explain how to do it and analyze how this affects the running time of step (iii).

(v) Finally, a horrifying idea: combining graphs and dynamic programming. Instead of building the above table, for all sensible combinations of i and s, why not explore the possibilities in a forward manner. Imagine a graph, whose vertices are the pairs (i, s) (position, speed) and whose edges represent legal game moves. Start at $(0, 1)$ and explore the graph, following the rules of the game. What graph traversal algorithm you will use to solve the problem? Explain how you can limit yourself only to the edges that are valid moves in the game. Estimate the worst-case running time of your algorithm. Do not forget to account for all costs. Aren't graphs wonderful?

Reminder: Algorithm descriptions must be written in careful English, pseudo-code, or any common programming language. Level of detail: enough to reconstruct what the algorithm actually does on any given input. Most of the time, you do not have to write actual code.

If you find yourself writing pages of code and/or English, you are doing something wrong. All the answers are relatively short.

Remember that if I ask for a *proof* of "In all possible cases, statement ZZZ holds." giving an example is not enough.