CMPS 102 — Winter Quarter 2017 – Homework 3

Solution to Problem 1 - Tokens

You're playtesting a new boardgame from one of your friends in the Game Design major, and they want your help to figure out the optimal strategy. The game has n rounds and in each round $1 \leq i \leq n$ you must acquire s_i tokens.
There are two possible ways to acquire tokens in round i:

(A) Spend r dollars per token, so $r * s_i$ in total, to acquire all tokens of round i.

(B) Spend C dollars to acquire the tokens of round i and of the next 3 rounds, independently of how many there are. (If you do this in round i, then your next choices concern rounds $i + 4$ and greater).

Example. Suppose $r = 1$, $C = 40$, and the sequence of s_i is $11, 9, 9, 12, 12, 12, 12, 9, 9, 11$. Then the cheapest way to acquire all the tokens is to choose action A for the first three rounds, then action B once, and then action A for the final three rounds.

Give a dynamic programming algorithm for finding the cheapest way to acquire all tokens, given the sequence $s_1, s_2, ..., s_n$. You must describe not only how you determine the value of the optimal sequence of actions, but also the sequence of actions itself.

$$\Big\{ xd \quad \text{if} lmao \tag{1}$$

Solution to Problem 2 - Monopoly

Another Game Design friend is working on a Monopoly spin-off with zoning laws! The board has n spaces, and the value of space i is $P(i)$. However, if you own space i you can't own any of spaces $i-2, i-1, i+1, i+2$.

(A) Give a dynamic programming algorithm for determining the maximum value of the spaces one can own. You do **not** need to output the corresponding set of spaces.

(B) Of course, in real Monopoly, the board wraps around, so let's take that into account as well. That is, assume that if you own space 1, you can not own spaces n or $n-1$ (besides spaces 2, 3). Solve this problem using DP (ideally, reusing your code for (a) (with tiny modifications) a few times).

Solution to Problem 3 - Happy Customers

You are working on n different projects, but in m hours you are going on vacation. Imagine that for each project i, you had a function f_i that told you how happy the people paying for project i would be if out of your n available hours you devoted $0 \leq x \leq m$ to project i. Imagine further that each such function has maximum value at most s_i, corresponding to the project being fully finished (and thus the clients being perfectly happy).

Give a dynamic programming for determining the most happiness you can generate for your clients by splitting your m hours among the n projects.

- Because of book-keeping rules, you can't spend fractional hours on a project.
- The functions f_i are non-decreasing, i.e., for every project i, if $t_1 \leq t_2$, then $f_i(t_i) \leq f_i(t_2)$
- The running time of your algorithm must be $O(n^a m^b s^c)$ for some fixed integers $a, b, c \geq 0$.

Solution to Problem 4 - The Food Bin

A person is shopping at a store that carries n different food items as packaged goods. A box of the i-th item will bring the shopper a benefit of b_i and cost p_i dollars, where b_i and p_i are integers. The shopper would like to receive maximum benefit, subject to the requirement that they can't spend more than P dollars in total. So, the problem is which boxes to take (they can take multiple boxes of the same item. This is known as the integral shopping problem.

In the fractional shopping problem, the same n items are sold in bulk, so the shopper can take as much or as little as they like of each food item, instead of having to buy a whole box or nothing.

1. Suppose that the items have the same order if we sort them by price from low to high, and if we sort them by benefit from high to low. Give an efficient algorithm to find an optimal solution to the integral shopping problem in this special case, and argue that your algorithm is correct.
   Hint: Describe the algorithm in English (1 line). Argue correctness in 3 lines.

2. Prove that the fractional version has the following property: the optimal solution can be found by making a series of locally optimal choices. Therefore, give a greedy algorithm to solve this version and prove it's correct.
   Hint: Ditto