Problem Set 8

All parts are due on April 27, 2019 at 6PM. Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Solutions should be submitted on the course website, and any code should be submitted for automated checking on `alg.mit.edu`.

Problem 8-1. [15 points] **Sunny Studies**

Tim the Beaver needs to study for exams, but it's getting warmer, and Tim wants to spend more time outside. Tim enjoys being outside more when the weather is warmer: specifically, if the temperature outside is t integer units above zero, Tim's happiness will increase by t after spending the day outside (with a decrease in happiness when t is negative). On each of the n days until finals, Tim will either study or play outside (never both on the same day). In order to stay on top of coursework, Tim resolves never to play outside more than two days in a row. Given a weather forecast estimating temperature for the next n days, describe an $O(n)$-time dynamic programming algorithm to determine which days Tim should study in order to increase happiness the most.

Problem 8-2. [15 points] **Selling Smoothies**

Little Tan Trotting Hood is a young entrepreneur. Her grandmother, Granny, makes delicious smoothies, and Little Tan wants to sell them at the local farmer's market. Granny has a diverse set of n bottles in her cupboard: one bottle b_i that holds exactly i milliliters of smoothie for each $i \in \{1, \dots, n\}$. Little Tan has conducted some market research to determine how much each bottle could sell for at the market when completely filled with smoothie. Granny has enough fruit to make at most $3n$ milliliters of smoothie. Given a revenue estimate r_i for each bottle b_i, describe an $O(n^2)$-time dynamic programming algorithm to determine which bottles to fill completely and sell in order maximize projected revenue at the market (partially filled bottles may not be sold).

Problem 8-3. [15 points] **Counting Courses**

Lance Weaklegs organizes the annual bike race in Gridville, a city laid out on an orthogonal grid of n streets, orthogonally crossing m avenues. Streets run East-West and are numbered sequentially $(1, \dots, n)$ from South to North, while avenues run North-South and are numbered sequentially $(1, \dots, m)$ from East to West. The race always occurs along some **North-Western course** through the city: a route along streets and avenues which: (1) begins at the intersection of 1st St. and 1st Ave., (2) ends at the intersection of n-th St. and m-th Ave, and (3) always travels North on avenues and West on streets (but may switch between them at intersections). However, some roads are closed for maintenance between certain intersections and may not be used in this year's race course. Lance wants to know how many different routes are possible. Given a list of all road closures, describe an $O(nm)$-time dynamic programming algorithm to count the number of possible North-Western courses through Gridville that avoid the road closures.

Problem 8-4. [15 points] **Capricious Contagion**

A strange and highly infectious virus has been spreading across the globe: it is deadly to some, yet others seem to be immune. CDC investigator Dr. Leah Boegist has found the DNA sequence of the virus to be string V. A **DNA sequence** is represented by a string of bases, with each base represented by a letter, either `A`, `C`, `G`, or `T`. Dr. Boegist observes that if the virus's DNA string V appears as a subsequence[1] of a patient's DNA sequence k or more times, then that patient will be immune to the virus and never get sick. Given V, integer k, and the DNA sequence S of a patient, describe an $O(|V||S|)$-time dynamic programming algorithm to determine whether the patient is in danger of ever getting sick. For example, if $V =$ `CAT` with $k = 2$, a patient with DNA $S =$ `TAGCATGTAG` will be immune from getting sick, because V appears twice as a subsequence of S.

Problem 8-5. [40 points] **Annoying Boxes**

You want to give a small gift to a friend, but not for nothing: they'll have to work for it! You will put the gift inside a box, and then fit that box inside another box, repeatedly as many times as you can. The gift will have **wrapping annoyance** k if it is nested inside k boxes. You have access to n rectangular boxes of known dimensions. The **dimensions** of box b_i is represented by a triple of integers (h_i, w_i, ℓ_i), corresponding to its height, width, and length respectively. One box can **fit inside** another box if the second box can be oriented so that all of its dimensions are strictly smaller than the dimensions of the first box. You may assume that the gift will fit inside any box. For example, a box with dimensions $(3, 6, 5)$ can fit inside a box with dimensions $(7, 4, 6)$ (after proper re-orientation), which would constitute a wrapping with annoyance 2.

(a) [15 points] Given a list B of triples representing box dimensions, describe an $O(|B|^2)$-time dynamic programming algorithm to determine a set of boxes that will maximize wrapping annoyance.

(b) [25 points] Write a Python function `annoy(B)` that implements your algorithm. The output of your algorithm should be a list of indices $A = [a_0, \ldots, a_{k-1}]$ such that box $B[a_{i-1}]$ can fit inside box $B[a_i]$ for all $i \in \{1, \ldots, k-1\}$, and k is the maximum wrapping annoyance possible using boxes from B. You **are allowed** to use Python's built-in sort functions if desired. You can download a code template containing some test cases from the website. Submit your code online at `alg.mit.edu`.

[1] Subsequences are not necessarily contiguous!