Problem Set 9

All parts are due on May 3, 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 9-1. [20 points] **The Temple of DPoom**

Kalifornia Kate has discovered a wall of writing in an ancient temple. She recognizes the writing as a sequence of **letters** from the Nacirema language. Some subsequences of letters form Nacirema **words**. Kate has a list of all w words in the Nacirema language, and notes that no word contains more than k letters. Words in Nacirema are written without spaces or punctuation, so a sequence of letters can be difficult to parse. A **valid parsing** of a sequence of letters is a partition of the letter sequence into a sequence of adjacent Nacirema words. Given a sequence $S = (s_1, \dots, s_n)$ of Nacirema letters, describe an $O((nk + w)k)$-time dynamic programming algorithm to count the valid parsings of S.

Problem 9-2. [20 points] **Making Myros**

The Myropean Myro is the currency of Myrope. Myrope prints Myro bills in k positive integer denominations: $D = \{d_1, \dots, d_k\}$. Auntie Mas is traveling to Myrope. To entertain herself while she is there, whenever she pays for anything, she wants to pay the exact amount with a limited number of Myro bills when possible. Given positive integers b and m, describe an $O(km)$-time dynamic programming algorithm to determine whether there exist b or fewer Myro bills that sum to exactly m Myros.

Problem 9-3. [20 points] **Linear Localities**

The thin island state of Long Linedon contains only a single road. Houses there have unique addresses that increase sequentially from 1 to n, from one end of the road to the other. Long Linedon has just completed a census, recording how many people p_i live at address i. The population has grown substantially since the last census, so Long Linedon needs to re-partition addresses into k new districts of roughly equal population. Districts must be **comprehensive**: every address must be in some district; and districts must be **contiguous**: for any triple of addresses a, b, c where $a < b < c$, if addresses a and c are in the same district, address b must also be in that district. The **population** of a proposed district is the number of people living in addresses contained in the district. Given the number of people living at each address and a positive integer k, describe an $O(n^2k)$-time dynamic programming algorithm to partition the addresses in Long Linedon into a set of k districts which maximizes the minimum population of any district in the set.

Problem 9-4. [20 points] **Wizard's Snakes and Ladders**

Twen Salamander challenges Frowndelwald to a game of "Wizard's Snakes and Ladders," a game played on a board containing n squares numbered from 1 to n. Each square i is marked with a positive integer number points p_i. At the beginning of the game:

- a **ladder** is placed between consecutive squares x and $x + 1$ for every $x \in \{1, \ldots, n - 1\}$;
- n snakes are randomly placed connecting between different pairs of squares;
- a positive integer **turn limit** k is fixed; and
- a token is placed at square 1.

The game has two players who alternate moving the token around the board. To make a move, a player will move the token from square a to some other square b where squares a and b are connected by either a ladder or a snake, after which that player will receive p_b points for moving the token to b. After $2k$ turns (k for each player), the player with the most points wins. Given an initial placement of snakes, the point values of all squares, and turn limit k, describe an $O(nk)$-time dynamic programming algorithm to determine whether Twen can force a win by playing first.

Problem 9-5. [20 points] **PineapplePhone**

Technology giant Pineapple makes a popular phone and wants to perform some optimizations. When a call comes in from a phone number, an internal dictionary is used to display the name of the caller. The dictionary is implemented using a binary search tree storing the phone's contacts keyed by phone numbers. The cost to lookup phone number p_i stored in the tree is equal to one plus the depth d_i of the node in which p_i is stored (so looking up the number stored at the root has cost one, and looking up in one of the root's children costs two). Employee Weave Stozniak wants to customize the binary search tree of each phone, to make more frequent phone numbers have a lower lookup cost than less frequent numbers. Every phone stores a list $C = \{c_1, \ldots, c_m\}$ of all calls ever received by the phone, and a list $P = \{p_1, \ldots, p_n\}$ of the unique phone numbers in the contacts list (you may assume that every call received in C is in the contacts list P). Let $m_i = |\{c_j \mid c_j \in C \text{ and } c_j = p_i\}|$ denote the number of calls received from phone number $p_i \in P$. Given P and C from a given phone, describe an $O(m + n^3)$-time dynamic programming algorithm to return a binary search tree that minimizes the expected lookup time for a phone number; specifically, return a binary search tree, from among all binary search trees on keys P, that minimizes $\sum_{p_i \in P}(1 + d_i)\frac{m_i}{m}$, where $\frac{m_i}{m}$ is the expected frequency of receiving a call from p_i.