Problem Set 9

All parts are due on April 26, 2018 at 11PM. 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 `py.mit.edu/6.006`.

Problem 9-1. [25 points] **Bidirectional Search**

(a) [5 points] **General Optimality:** Breadth-first search determines a shortest path from s to t in an unweighted graph $G = (V, E)$ in $O(|V| + |E|)$ time. Show that the running time of this algorithm is asymptotically optimal applied to general graphs, by describing an unweighted undirected graph $G = (V, E)$ containing vertices s and t for which the shortest path from s to t has size $\Omega(|V| + |E|)$.

(b) [10 points] **Specific Improvement:** For many graphs, single source breadth-first search can be done faster by making the search **bidirectional**: perform a breadth-first search separately from s and t, alternating the exploration of level i from s, then level i from t, then level $i + 1$ from s, etc., until the frontier levels from the two searches intersect, returning the concatenation of the two shortest paths found[1]. Consider the family of unweighted undirected graphs having bounded degree b, where a shortest path from vertices s to t is known to contain d edges. Describe a graph from this family for which breadth-first search from s takes at least $\Omega((b-1)^d)$ time, while bidirectional breadth-first search from s and t takes at most $O(b^{d/2})$ time.

(c) [10 points] **Bidirectional Dijkstra:** A similar algorithm can speed up Dijkstra on graphs containing non-negative edge weights, by running Dijkstra separately from s and t until their 'frontiers' intersect. Each Dijkstra search maintains a priority queue (e.g., Q_s) initially containing all vertices, and the set of vertices already extracted from the queue (e.g., $V \setminus Q_s$) represents the frontier of that search. To interleave their execution, extract and process a vertex from a search queue if its minimum path weight estimate to its source is smallest in both queues. Naively one might think the same stopping condition as bidirectional BFS might work here: stop the search as soon as the same vertex appears in the frontier of both searches, returning the concatenation of the two shortest paths found; however this stopping condition does not always return a shortest path[2]. Describe a weighted undirected graph on five vertices such that running bidirectional Dijkstra between two of the vertices using the above stopping condition returns a path that does not have minimum weight.

[1] An interesting exercise is to prove that this path is indeed a shortest path.

[2] A correct stopping condition for bidirectional Dijkstra is to: maintain a vertex v whose minimum weight path estimates to s and t sum to the smallest value $\mu = d_s(v) + d_t(v)$, and stop the search when μ is no larger than the sum of the smallest minimum path estimates remaining in both searches' priority queues.

Problem 9-2. [20 points] **Katalan**

Kat A. Lan is taking a class in combinatorics. Having already learning about binary trees in 6.006, she is intrigued to learn that something called the nth **Catalan number** C_n corresponds to the number of rooted binary trees containing $n-1$ non-leaf vertices, where every non-leaf vertex has exactly two children. There are many ways to compute Catalan numbers, but Kat is intrigued by the following recursive definition:

$$C_1 = 1 \qquad C_n = \sum_{i=1}^{n-1} C_i C_{n-i}.$$

Kat writes the following code to compute Catalan numbers according to this definition.

```
def catalan(n):
    if n == 1:
        return 1
    out = 0
    for i in range(1, n):
        out += catalan(i) * catalan(n - i)
    return out
```

(a) [5 points] Is Kat's code correct? Does Kat's code run in polynomial time?

(b) [5 points] Describe a directed graph of computational dependencies of the first n Catalan numbers based on the recursive definition above. How many vertices and edges are in this graph?

(c) [10 points] Improve Kat's code by writing a Python function to compute the nth Catalan number in $O(n^2)$ time. Note there are faster ways to compute Catalan numbers; we are looking for a **dynamic programming** solution based on the **provided recursive definition**. Your code may use either a bottom up or top down approach. Include your code in your PDF submission. Your code should not be more than 15 lines long (our solution is under 10 lines).

Problem 9-3. [20 points] **Dynamic Consulting**

Solve each of the following problems using **dynamic programming**. Specifically: define subproblems, relate them recursively, show that subproblems are acyclic, state base cases, show how to compute the solution, and show that your dynamic program achieves the requested running time.

(a) [10 points] **Future Investing:** Tiffany Bannen stumbles upon a lottery chart dropped by a time traveler from the future, which lists winning lottery numbers and cash payout for the next n days. Tiffany is excited to use this information to make money, but is worried that if she plays winning numbers every day, the organizers of the lottery will get suspicious. As such, she decides to limit herself to playing the lottery at most twice in any seven day period. Describe a $O(n)$ time algorithm to determine the maximum amount of lottery winnings Tiff can get in the next n days, subject to playing the lottery at most twice in any seven day period.

(b) [10 points] **Light Show:** Glark Criswold recently purchased a programmable string of n LEDs to accentuate his house for the holidays. Each LED on the string can be independently assigned a different color at high frequency to simulate elaborate visual animations. The programmable interface of the LED string allows a user to update any contiguous subsequence of LEDs along the string with a set of new colors values, ignoring any updates that are out of the range of the LED strip . However, because of issues concerning electrical impedance and caching, modifying contiguous subsequences of different length takes different amounts of time. For example, updating the color of a single LED might take $2\mu s$, updating a subsequence of ten LEDs might take $30\mu s$, and updating a subsequence of twenty LEDs might take $20\mu s$ (these values are not necessarily proportional to the number of updated LEDs). Given the current color a_i and desired color b_i of LED i, for each of the n LEDs, and the time t_k of performing an update on any contiguous subsequence of k LEDs, describe an $O(n^2)$ time algorithm to determine the minimum time to update each a_i to b_i. (Note that if $a_i = b_i$, LED i does not need to be updated.)

Problem 9-4. [35 points] **Choosing Prizes**

Alyssa P. Hacker just won the game show, "Who Wants to Be A Theoretical Computer Scientist!" For winning the contest, she is shown a row of prizes. Alyssa is allowed to choose and take home any subset of the prizes, so long as no two chosen prizes are next to each other in the row. Each prize is displayed with a monetary value, so naturally Alyssa wants to maximize the total value of the prizes she selects. For instance, if the prize values in the row are: [14, 30, 27, 4, 5, 15, 1], Alyssa should pick the prizes worth 14, 27, and 15, as this is a subset with the maximum total.

(a) [10 points] Design a linear time algorithm to determine which prizes Alyssa should choose to maximize the total value of the prizes she selects.

(b) [25 points] Write the Python function `choose_prizes(prize_values)` that implements your algorithm from part (a). The input `prize_values` is a list of length n containing the prize values, which are positive integers. Your function should return a list of the indices between 0 and $n-1$ inclusive not containing any consecutive values, corresponding to the prizes that Alyssa should select. Submit your code online at `py.mit.edu/6.006`.