Recap of this week’s lectures:

- Dynamic programming
- Single-source shortest path algorithms: Dijkstra and Bellman-Ford
- All-pairs shortest path algorithms: Floyd-Warshall and Matrix-Mult
- Subset DP for TSP

1. Bottleneck Paths. Given a directed graph $G$, suppose each edge has a non-negative capacity $c_e$. Given a directed path from $s$ to $t$, the bottleneck edge of this path is the min-capacity edge on it. The $s$-$t$ bottleneck path asks for such a path whose bottleneck edge capacity is as large as possible.

Show how to modify Dijkstra’s algorithm to solve this problem in $O(m \log n)$ time (or $O(m + n \log n)$ time using Fibonacci heaps).

2. Johnson’s Algorithm. We did not get a chance to discuss Johnson’s algorithm for APSP, we do that now. (Details in notes, Section 4.3 of Lecture 8).

(a) If the edge-weights are non-negative, running Dijkstra from each node takes $n \cdot O(m + n \log n)$ time. Much faster than F-W if the graph is sparse, i.e., $m \ll n^2$.

(b) Suppose we assign an “offset” $\Phi_v$ to each vertex $v$, and define the offset edge-length of edge $(u, v)$ to be $\ell_{uv}' := \Phi_u + \ell_{uv} - \Phi_v$. Show that a $s$-$t$ path is a shortest path with respect to lengths $\ell$ if and only if it is a shortest path with respect to lengths $\ell'$.

(c) Call an offset “feasible” if $\ell_{u,v}' \geq 0$ for all edges, even if $\ell$ could have been negative. Suppose there is a vertex $x$ that has finite distance to every other vertex. Show that $\Phi_v := \text{dist}(x, v)$ is a feasible offset.

(d) Infer that you can compute APSP by running Bellman-Ford once (to compute the feasible offset) and then $n$ Dijkstras.
3. **Coloring Dynamically.** Given a graph $G = (V, E)$, a (proper) $k$-coloring is a coloring of the vertices of the graph using $k$ colors, so that the endpoints of each edge get distinct colors. An equivalent definition is that the vertices having any single color form an independent set.

This problem is NP-hard for $k \geq 3$, so we explore fast exponential-time algorithms.

(a) Show that the problem of 2-coloring a graph can be solved in linear time.

(b) The naive algorithm to $k$-color a graph takes $k^n$ time. Give an algorithm that runs in time $O(k3^n)$. [A simpler bound is $O(k4^n).]

4. **Bin-Packing.** You are given a collection of $n$ items, each item has size $s_i \in [0, 1]$. You have many bins, each of unit size, and you want to pack the $n$ items into as few bins as possible. (Each bin can take a subset of items, whose total size is at most 1.)

Show that you can solve this problem in time $O(n3^n)$. (Hint: subset DP.)