15-451/651 Algorithms, Spring 2019
Recitation #5 Worksheet

Recap of this week's lectures:

- Dynamic programming: top-down and bottom-up
- Examples: Knapsack, Independent set in trees, LCS, etc.
- Single-source shortest path algorithms: Dijkstra and Bellman-Ford
- All-pairs shortest path algorithms: Floyd-Warshall and Matrix-Mult
- Subset DP for TSP

---

1. **Off-Line Stock Market Problem** You're given a sequence of stock prices $[p_1, p_2, \ldots, p_n]$. You want to find the maximum profit that you could have made on the stock in hindsight. In other words, you want to find i and j with $1 \le i \le j \le n$ such that $p_j - p_i$ is maximal. Your algorithm should run in $O(n)$ time.

2. **Longest Increasing Subseqeuence:** Given an array A of n integers like `[7 2 5 3 4 6 9]`, find the longest subsequence that's in increasing order (in this case, it would be `2 3 4 6 9`). Give a dynamic-programming algorithm that runs in time $O(n^2)$ to solve this problem.

   (a) To keep things simple, first let's say you just need to output the *length* of the longest-increasing subsequence. E.g., in the above case, the length is 5.

   (b) Now extend your solution to actually find the LIS.

3. **Making Change:** You are given denominations $v_1, v_2, \ldots, v_n$ (all integers) of the various kinds of currency you have. (Say $v_1 = 1$, so you can make change for any integer amount $C \geq 1$.) Given C, give a dynamic programming solution which makes change for C with the fewest bills possible.

   (Again, as a first stab, compute the number of bills required, and then extend the solution to output the number of bills of each denomination needed.)

4. **Making Change (Part II):** Now suppose you have only one bill of each denomination i. Given C, give a dynamic programming solution which makes change for C using the fewest bills, using no more than one bill of each denomination i (or says this is not possible).

5. **Making Change (Part III):** Can you solve the problem if you have ℓ_i bills of denomination i?

6. **Balanced Partition.** You have a set of n integers each in the range $0, \ldots, K$. In time $O(n^2 K)$, partition these integers into two subsets such that you minimize $|S_1 - S_2|$, where S_1 and S_2 denote the sums of the elements in each of the two subsets.

7. **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).

8. **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" Φ_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 ℓ if and only if it is a shortest path with respect to lengths ℓ'.

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

   (d) Infer that you can compute APSP by running Bellman-Ford once (to compute the feasible offset) and then n Dijktras.

9. **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 $O(k4^n)$.]

10. **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.)