CS 256: Algorithm Design and Analysis

Problem Set 6 (due 04/16/26)

Instructor: Sam McCauley

Note on Dynamic Programs. For full credit on a dynamic program, you must clearly state the following parts.

(a) *Subproblem definition:* your subproblem must have an optimal substructure.
(b) *Recurrence:* how should the next subproblem be computed using the previous ones? This is the core of your algorithm and its correctness. A less ideal alternative to a recurrence is clear pseudocode for the final iterative dynamic-programming algorithm.
(c) *Base case(s):* you need to start somewhere!
(d) *Final output:* in terms of your subproblem.
(e) *Memoization data structure:* this is often obvious but should not be skipped.
(f) *Evaluation order:* describes the dependencies between the subproblems.
(g) *Time and space analysis.*

For this assignment, these are sufficient to argue correctness (that is to say, if you explain why the above parts are correct, that's sufficient to show that your algorithm works properly).

Problem 1. (Erickson 3.6)

A shuffle of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of $X$and Yin the same order.For example, the string BANANAANANAS is a shuffle of the strings BANANA and ANANAS in several different ways.

$$\mathsf{BANANA}_{\mathsf{ANANAS}} \qquad \mathsf{BAN}_{\mathsf{ANA}}\mathsf{ANA}_{\mathsf{NAS}} \qquad \mathsf{B}_{\mathsf{AN}}\mathsf{AN}_{\mathsf{A}}\mathsf{A}_{\mathsf{NA}}\mathsf{NA}_{\mathsf{S}}$$

Similarly, the strings PRODGYRNAMAMMIINCG and DYPRONGARMAMMICING are both shuffles of DYNAMIC and PROGRAMMING:

$$\mathsf{PRO}_{\mathsf{D}}\mathsf{G}_{\mathsf{Y}}\mathsf{R}_{\mathsf{NAM}}\mathsf{AMMI}_{\mathsf{I}}\mathsf{N}_{\mathsf{C}}\mathsf{G} \qquad {}_{\mathsf{DY}}\mathsf{PRO}_{\mathsf{N}}\mathsf{G}_{\mathsf{A}}\mathsf{R}_{\mathsf{M}}\mathsf{AMM}_{\mathsf{IC}}\mathsf{ING}$$

Given three strings $A[1..m]$, $B[1..n]$, and $C[1..m+n]$, describe and analyze a dynamic programming algorithm to determine whether C is a shuffle of A and B. **Remember to fill in all parts of the recipe for a dynamic program** (you can find a reminder of them above).

Solution. $\square$

Problem 2. (From Steve Skiena's *Algorithm Design Manual*) Consider the problem of storing n books on shelves in a library. The order of the books is fixed by the cataloging system and so cannot be rearraged. Let book b_i have thickness t_i and height h_i, for $1 \leq i \leq n$. Let the length of each bookshelf at this library be L. Suppose we have the freedom to adjust the height of each shelf to fit the tallest book on it. The cost of a particular layout is the sum, over each shelf, of the height of the largest book on that shelf. (So if shelf 1 has books with heights $(1, 5, 3)$ and shelf 2 has books with heights $(2, 4)$, the total cost is $5 + 4 = 9$.)

(a) Give an example to show that the greedy algorithm of stuffing each shelf as full as possible (that is, fill the first shelf with as many books as possible until book b_i does not fit, and then repeat the same process on subsequent shelves) does not always give the minimum overall height.

(b) Give a dynamic programming algorithm that computes the height of the optimal arrangement, and analyze its time and space complexity.

> **Hint.** We have done a similar example in class with a different cost function and constraints.

Solution. □

Problem 3. Two friends Rosa and Beth are planning a hike to the top of a mountain. There are n lookout points situated at varying points on the mountain (including one at the top), and an extensive network of trails connecting these lookout points.[1]

Rosa and Beth want to find the shortest path from the base of the mountain to the top of the mountain under two constraints. First, Rosa and Beth want to (between the two of them) visit all n lookout points: any lookout point i should either be on Rosa's route or Beth's route. Second, neither wants to backtrack: if Rosa visits lookout point i, Rosa's next lookout point should be at a higher elevation. Similarly, when Beth visits some lookout point j, Beth's next lookout point should be at a higher elevation.

Assume that the n lookout points are given in sorted order of elevation and that there are no ties (so lookout point i is higher than lookout point k when $i > k$).

You are given the following input: for each pair of lookout points x, y, you are given the distance $d(x, y)$ to hike from lookout point x to lookout point y. Give an algorithm to find the shortest *total* path length (i.e. the sum of Rosa's path length and Beth's path length) through the mountain such that (1) Rosa or Beth visit all lookout points, and (2) the path Rosa takes, and the path Beth takes, each visit lookout points in increasing order. Both paths start at lookout point 1, and end at lookout point n (lookout point n is at the top of the mountain).[2]

Give an $O(n^2)$ dynamic programming algorithm to find the optimal pair of paths, minimizing the total distance hiked in sum by Rosa and Beth (In the "time and space" section please explain why your algorithm is $O(n^2)$.)

Solution.

(a) **Subproblem definition:**

Hint. Use the following subproblem definition. You do not need to create your own subproblem for this problem. (If you want to change this that's OK—but there is a correct answer that uses this subproblem definition.)

Entry (i, j) in the dynamic programming table represents two paths such that Rosa's path is from 1 to i; Beth's path is from 1 to j, and every vertex from 1 to $\max\{i, j\}$ is visited by one of the two paths.

Let $S(i, j)$ be the smallest total length (sum of the two paths) of Rosa and Beth's path satisfying these constraints.

(We do not use $S(i, i)$.)

(b) **Recurrence:**

[1] Note that we don't use the specific trail layout in this problem: we just use the distance between any two lookout points. Basically: assume there is some path from any lookout point to any other.

[2] This means that both Rosa and Beth visit lookout points 1 and n. Any other lookout point will only be visited by one of them.

> **Hint.** First, let's say that either $i > j + 1$, or $j > i + 1$. What is the recurrence for $S(i, j)$? (This recurrence is unusually short.)
>
> Then, let's say that $i = j + 1$ or $j = i + 1$. What is the recurrence for $S(i, j)$?

(c) **Base case(s):**

(d) **Final output:**

(e) **Memoization data structure:**

(f) **Evaluation order:**

(g) **Time and space analysis**

> **Hint.** For the time analysis, treat the total time of the two cases in your recurrence separately.

$\square$

Problem 4 (From "Algorithms Illuminated" by Roughgarden)**.**

(a) In the Bellman-Ford algorithm, we update the shortest-path estimate to each vertex in each iteration. Let $M[v, i]$ represent the shortest known distance to vertex v after the i-th iteration.

Suppose that, at some iteration k, no vertex's distance changes in the next iteration. That is, $M[v, k + 1] = M[v, k]$ for *all* vertices v.

Explain why this means that the algorithm has "stabilized" and why we can safely stop early. In other words, why will the distances remain unchanged in all future iterations (i.e., $M[v, i] = M[v, k]$ for all $i > k$)? You do not need to give a formal proof, but you should explain explicitly why no iterations are required—your answer should reference how the Bellman-Ford algorithm works.

Solution. □

(b) Now consider stabilization on a *per-vertex* basis.

Suppose that for some vertex v and some iteration k, the distance to v does not change in the next iteration; that is, $M[v, k + 1] = M[v, k]$.

Does this imply that the distance to v will remain unchanged in all subsequent iterations (i.e., $M[v, i] = M[v, k]$ for all $i \geq k$)?

Prove this statement, or provide a counterexample.

Solution. □