Fundamental Algorithms
Courant, Fall 2017

Homework 7

Instructor: Amir Shpilka
Due Date: Wed Nov 1, 11:55pm

1. (Don't be greedy)

   15.1-2 (a) (3 points) Show, by means of a counterexample, that the following "greedy" strategy does not always determine an optimal way to cut rods. Define the density of a rod of length i to be p_i/i, that is, its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where $1 \leq i \leq n$, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length $n-i$.

   15.3-4 (b) (3 points) As stated, in dynamic programming we first solve the subproblems and then choose which of them to use in an optimal solution to the problem. Professor Betsy claims that we do not always need to solve all the subproblems in order to find an optimal solution. She suggests that we can find an optimal solution to the matrix-chain multiplication problem by always choosing the matrix A_k at which to split the subproduct $A_i A_{i+1} \cdots A_j$ (by selecting k to minimize the quantity $p_{i-1}p_k p_j$) *before* solving the subproblems. Find an instance of the matrix-chain multiplication problem for which this greedy approach yields a suboptimal solution.

2. (a) (4 points) Given a tree (not necessarily binary), show how to calculate the height of the subtree rooted at every node. Your algorithm should take time $O(n)$ where n is the number of nodes in the tree.

   15-1 (b) (5 points) Suppose that we are given a directed acyclic graph $G = (V, E)$ with real-valued edge weights and two distinguished vertices s and t (see CLRS Sect. B.4 for definitions). You can assume that each node u stores an array `edges`, that contains pairs (v, k) that encodes an edge from u to v with weight k. Describe a dynamic-programming approach for finding a longest weighted simple path from s to t. What does the subproblem graph look like? What is the efficiency of your algorithm?

15-2 3. (5 points) A palindrome is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `civic`, `racecar`, and `aibohphobia` (fear of palindromes).

   Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input `character`, your algorithm should return `carac`. What is the running time of your algorithm?

4. Suppose you are given an array $A[1, \dots, n]$ of numbers, which may be positive, negative, or zero.

   (a) (4 points) Let $S_{i,j}$ denote $A[i] + A[i+1] + \cdots + A[j]$. Use dynamic programming to give an $O(n^2)$ algorithm to compute $S_{i,j}$ for all $1 \leq i \leq j \leq n$, and hence compute $\max_{i,j} S_{i,j}$.

   (b) (5 points) Let $L[j]$ denotes $\max_i S_{i,j}$. Give a recurrence relation for $L[j]$ in terms of $L[1, \dots, j-1]$. Use your recurrence relation to give an $O(n)$ time dynamic programming algorithm to compute $L[1 \dots n]$, and hence compute $\max_{i,j} S_{i,j}$.

   (c) (2 points) Consider the naive recursive algorithms (without memorization, like `Cut-Rod` from class) based on the recurrence relations to compute the answers to part(a) and part(b). Will both running times stay at $O(n^2)$ and $O(n)$, respectively, only one of them (which one?), or none?

   (d) (4 points) Suggest appropriate modifications to your algorithm in part (b) to give an $O(n)$ algorithm to compute $\max_{i,j} P_{i,j}$, where $P_{i,j} = A[i] \cdot A[i+1] \cdots A[j]$. Assume that multiplication of any two numbers takes $O(1)$ time.

5. Let $X[1 \dots m]$ and $Y[1 \dots n]$ be two given arrays. A common supersequence of X and Y is an array $Z[1 \dots k]$ such that X and Y are both subsequences of $Z[1 \dots k]$. Your goal is to find the *shortest* common super-sequence (SCS) Z of X and Y, solving the following sub-problems.

   (a) (4 points) First, concentrate on finding only the length k of Z. Proceeding similarly to the longest common subsequence problem, define the appropriate array $M[0 \dots m, 0 \dots n]$ (in English), and write the key recurrence equation to recursively compute the values $M[i, j]$ depending on some relation between $X[i]$ and $Y[j]$. Do not forget to explicitly write the base cases $M[0, j]$ and $M[i, 0]$, where $1 \leq i \leq m, 1 \leq j \leq n$.

   (b) (5 points) Translate this recurrence equation into an explicit bottom-up $O(mn)$ time algorithm that computes the length of the shortest common supersequence of X and Y.

   (c) (5 points) Find the SCS of $X = BARRACUDA$ and $Y = ABRACADABRA$. (Notice, you need to find the actual SCS, not only its length.)

   (d) (6 points) Show that the length k of the array Z computed in part (a) satisfies the equation $k = m + n - \ell$, where ℓ is the length of the longest common *subsequence* of X and Y.

   Hint: Use the recurrence equation in part (a), then combine it with a similar recurrence equation for the LCS, and then use induction. There the following identity is very handy: $\min(a, b) + \max(a, b) = a + b$.