CS 341, Fall 2020 A. Lubiw, A. Storjohan

ASSIGNMENT 4

DUE: Wednesday October 21, 5 PM. DO NOT COPY. ACKNOWLEDGE YOUR SOURCES.

Please read `http://www.student.cs.uwaterloo.ca/~cs341` for general instructions and policies.

Exercises. The following exercises are for practice only. You may ask about them in office hours. Do not hand them in.

1. Give a dynamic programming algorithm to find the longest palindrome that is a subsequence of a given string. Recall that a *palindrome* is a word that reads the same forwards and backwards, like "racecar" or "kayak". For example, on input "character", your algorithm should output "carac". Notes: (1) a *subsequence* means that the letters do not have to be consecutive; (2) the palindrome you find does not have to be an English word.

   Hint: pull off a letter from the front, or the back, or both to get three smaller problems.

Problems. To be handed in.

1. [10 marks] **Dynamic programming to chop a lecture video into smaller segments.** Some sites have a limit on the time or space of videos, which means that video lectures need to be broken into smaller segments. How should we choose the break points? One issue is that two consecutive slides might be closely related so putting a segment break between them would be bad. This question is about breaking the slides into segments to minimize the bad breaks.

   Suppose a lecture video consists of n slides with narration and suppose that the ith slide takes t_i minutes, $i = 1, \dots, n$. Suppose the time limit on a video segment is T minutes, and assume that $t_i \leq T$ for all i. Suppose that introducing a segment break between slides i and $i+1$ has a "badness factor" of $b_i, i = 1, \dots, n-1$. Give a dynamic programming algorithm to break the sequence of slides into segments, such that each segment takes at most T minutes, and such that the sum of the badness of the breaks is minimized.

   In notation, we want a list of breaks, $(i_1, i_2, \dots, i_k)$ with $1 \leq i_1 < i_2 \cdots < i_k \leq n$, to give $k+1$ segments: $[1..i_1]$, $[(i_1+1)..i_2]$, $\dots$, $[(i_k+1)..n]$. The condition is that each segment must take time $\leq T$. The goal is to minimize the sum $\sum\{b_{i_j} : j = 1, \dots, k\}$.

   Hint. A solution with runtime $O(n^2)$ is possible. Slower solutions get part marks.

   Example. Consider $n = 3$, $T = 10$, with times $t = (6, 1, 4)$ and badness factors $b = (8, 9)$. Since the sum of the times is 11, we need more than one segment. There are three possible lists of breaks to cut into segments of time ≤ 10: $(1, 2)$ (giving three segments each with one slide); or (1); or (2). The corresponding badness sums are $8 + 9 = 17$, 8, and 9, respectively. Thus the optimum solution is to put a break after slide 1, giving two sequences, the first taking 6 minutes, and the second taking $1 + 4 = 5$ minutes.

   (a) [0 marks, enrichment only] Prove that if the b_i's are all equal to 1, then there is a greedy algorithm to solve the problem, and the solution will minimize the number of segments.

(b) [7 marks] Give a dynamic programming algorithm to compute the minimum badness. You must fill in the whole DP array in an iterative bottom-up fashion—don't use memoization. Your solution must consist of:

   i. A description of the subproblems you will solve and the order in which you will solve them.
   ii. A formula for solving one subproblem in terms of previously solved subproblems, and a justification of correctness.
   iii. Pseudocode of your algorithm.
   iv. An analysis of run time using big oh notation.

(c) [3 marks] Using the solutions to subproblems you found in part (b), give an algorithm to compute the actual list of breaks for an optimum solution.

(d) [0 marks, enrichment only] Show how to modify the b_i's so that the algorithm produces a solution that first minimizes the number of segments, and then minimizes the badness.

2. [10 marks] **Dynamic programming for two suitcases.** Suppose you are moving for a co-op job, and want to choose which of your belongings to take. You have two suitcases. Suitcase 1 will be weighed and there is a limit of W kilos. Suitcase 2 won't be weighed, but it has a volume limit of V. Suitcase 1 has no volume limit and suitcase 2 has no weight limit. You have n items, $1, 2, \ldots, n$. Item i has weight $w_i \in \mathbb{N}$, volume $v_i \in \mathbb{N}$, and benefit b_i if you take it with you. You can leave items behind.

The goal is to find the maximum benefit by choosing items for the two suitcases while observing weight limit W for suitcase 1 and volume limit V for suitcase 2.

In notation, you want sets $S_1, S_2 \subseteq \{1, 2, \ldots, n\}$ such that

1. $\sum_{i \in S_1} w_i \leq W$,
2. $\sum_{i \in S_2} v_i \leq V$, and
3. $B = \sum_{i \in S_1 \cup S_2} b_i$ is maximized.

Give a dynamic programming algorithm to find the maximum benefit B. You must fill in the whole DP array in an iterative bottom-up fashion—don't use memoization. Your algorithm does not need to find the sets S_1 and S_2. Your solution must consist of:

(a) A description of the subproblems you will solve and the order in which you will solve them.
(b) A formula for solving one subproblem in terms of previously solved subproblems, and a justification of correctness.
(c) Pseudocode of your algorithm.
(d) An analysis of run time using big oh notation.
(e) Is your algorithm a polynomial-time algorithm? Why or why not?

Programming Assignment 1 will ask you to implement your algorithm. More details will be available soon.

Challenge Questions. This is for fun and enrichment only. Do not hand it in.
See questions 1(a) and (d).