Homework 4. Due February 22

CS180: Algorithms and Complexity
Winter 2017

GUIDELINES:

- Upload your assignments to Gradescope by 6:59 PM.
- Follow the instructions mentioned on the course webpage for uploading to Gradescope very carefully (including starting each problem on a new page and matching the pages with the assignments); this makes it easy and smooth for everyone. As the guidelines are simple enough, bad uploads will not be graded.
- You may use results or algorithms proved in class without proofs as long as you state them clearly.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly. Homework is a stepping stone for exams; keep in mind that reasonable partial credit will be awarded and trying the problems will help you a lot for the exams.

1. When their respective sport is not in season, UCLA's student-athletes are very involved in their community, helping people and spreading goodwill for the school. Unfortunately, NCAA regulations limit each student-athlete to at most one community service project per quarter, so the athletic department is not always able to help every deserving charity. For the upcoming quarter, we have S student-athletes who want to volunteer their time, and B buses to help get them between campus and the location of their volunteering. There are F projects under consideration; project i requires s_i student-athletes and b_i buses to accomplish, and will generate $g_i > 0$ units of goodwill for the university.

   Use dynamic programming to produce an algorithm to determine which projects the athletic department should undertake to maximize goodwill generated. For full-credit, your algorithm should run in time $O(SBF)$ but you don't have to prove its correctness or analyze the time complexity. [.75 points]

   (Hint: This is similar to the knapsack problem we did in class with the caveat that we have two 'constraints': number of students and number of buses (just as the weight was a constraint in knapsack). Can you use this to choose the right subproblems? Also, note that you have to determine the best set of projects and not just the best value. For this part, you can design an algorithm similar to the one we used for knapsack based on working back from the recurrence you get.)

2. You have a knapsack of total weight capacity W and there are n items with weights $w_1, \ldots, w_n$ respectively. Give an algorithm to compute the number of different subsets that you can pack safely into the knapsack. In other words, given integers $w_1, \ldots, w_n, W$ as input, give an algorithm to compute the number of different subsets $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$. For full-credit, your algorithm should run in time $O(nW)$ but you don't have to prove its correctness or analyze the time complexity. [.75 points]

   (Hint: This is similar to the knapsack problem, but instead of looking for the optimal subset you are looking for the total number of *feasible* subsets. Nevertheless, you can use build on the same basic approach.)

3. Given two strings $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$, let $deleteScore(X, Y)$ be the least number of characters you have to delete from X, Y so that you get two same strings. For example, if $X = goodman$ and $Y = goldmann$, then $deleteScore(X, Y) = 3$ (you can delete 'o' from X, 'l' and one 'n' from Y to get the same string - "godman").

   Give an algorithm that given two strings X, Y computes $deleteScore(X, Y)$. For full-credit, your algorithm should run in polynomial time. [.75 points]

   (Example: On input $X = goodman$, $Y = goldmann$ your output should be 3.)

   (Hint: One possible approach is to use subproblems and a recurrence relation that is similar in spirit to something we did in class.)

4* There are four types of brackets: (,), <, and >. We define what it means for a string made up of these four characters to be *well-nested* in the following way:

   (a) The empty string is well-nested.

   (b) If A is well-nested, then so are <A> and (A).

   (c) If S, T are both well-nested, then so is their concatenation ST.

   For example, (), < () >, ((< >)), () <> are all well-nested. Meanwhile, (, <>),)(, (<)>, and (<(> are not well-nested.

   Devise an algorithm that takes as input a string $s = (s_1, s_2, ..., s_n)$ of length n made up of these four types of characters. The output should be the length of the shortest well-nested string that contains s as a subsequence. For example, if (<(> is the input, then the answer is 6; a shortest string containing s as a subsequence is ()<()>. For full-credit, your algorithm should run in time $O(n^3)$ but you don't have to prove its correctness or analyze the time complexity. [.75 points]

   (Hint: For this problem, try to use sub-problems that solve the problem for substrings of the original s: $(s_i, s_{i+1}, \ldots, s_j)$ for all $i < j$.)

ADDITIONAL PROBLEMS. DO NOT turn in answers for the following problems - they are meant for your curiosity and understanding.

1. Chapter 6, problems 1, 2, 3, 4, 5, 11, 15, 20, 26, and 27. from textbook [KT].

2. Problems 6.7, 6.8, 6.11, 6.18, 6.25, 6.26 from Chapter 6 of [DPV].