Homework 4. Due May 14

CS180: Algorithms and Complexity
Spring 2018

GUIDELINES:

- Upload your assignments to Gradescope by 5:59 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 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. Problem 5.1 from [DPV]. You should break ties between edges of same weight (if needed) in lexicographic order. [.75 points]

   (In second part of part(c), "For each edge in this sequence, give a cut that justifies its addition." means to find a cut for which the edge being added is an edge of minimum weight crossing the cut.)

2. You are given a connected graph G with n vertices and m edges, and a minimum spanning tree T of the graph. Suppose one of the edge weights $c(e)$ for an edge $e \in T$ is updated. Give an algorithm that runs in time $O(m)$ to test if T still remains the minimum spanning tree of the graph. You may assume that all edge weights are distinct both before and after the update. Explain why your algorithm runs in $O(m)$ time and is correct. [.75 points]

   (Hint: Consider the cut obtained by deleting e from T. Note that as you are only allowed $O(m)$ time, you cannot recompute a MST in the new graph from scratch. Also, make sure you specify your algorithm fully.)

3. 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.)

4. 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.)

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

1. Problem 5.5, 5.6, 5.7 from Chapter 5 of [DPV].
2. Problems 4.1, 4.2, 4.3, 4.14 from textbook [KT].
3. You can try to come up with variants of problems we did in class (e.g., interval scheduling, weighted interval scheduling, knapsack, etc.,) and see if they can be solved by dynamic programming. If you think you have interesting ones, let me know!

   For example, what if you have three knapsacks (or constraints: like weight, height etc.,) or multiple copies of items.
4. Problems 6.10*, 6.16, 6.17, 6.18, 6.25 from Chapter 6 of [DPV].
5. Chapter 6, problems 1, 2, 3, 4, 11 from textbook [KT].