CS7800: Advanced Algorithms. Fall 2016
Homework 3

Instructor: Jonathan Ullman, TA: Mehraneh Liaee

Due Friday, October 21 at 11:59pm
(Email to m.liaee2050+CS7800@gmail.com)

- In order to make this HW less time consuming, some of the problems only ask for the part of the algorithm that I consider most interesting. For these problems, I have also included questions, marked in red, that you *do not* need to answer in your submission. Even though you do not need to submit answers, it's important for the exams that you are able to answer these questions, so please make sure that you are confident you could answer them if I asked.
- You must type your solutions using LaTeX. Please submit both the source and PDF files using the naming conventions `lastname_hw3.tex` and `lastname_hw3.pdf`.
- **Please put your name somewhere on the first page of your submission.**
- Strive for clarity and conciseness in your solutions, emphasizing the main ideas over low-level details. I recommend looking at the introduction in Jeff Erickson's textbook for advice on writing up solutions to algorithms problems.
- Do not share written solutions, and remember to cite all collaborators and sources of ideas. Sharing written solutions, and getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

Dynamic Programming

Problem 1 (Segmented Least Squares with Exactly K Segments, 15 points)**.**

In class we considered a version of the segmented least squares problem where there are n input points $P = \{p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)\}$ and the goal was to produce a piecewise-linear function to minimize $E + Ck$ where E is the error and k is the number of segments used by our solution. Suppose instead we want to find a piecewise-linear function with *exactly* K segments. That is, we minimize E subject to the constraint $k = K$. For simplicity, assume that all of the points have a distinct x coordinate.

Formulate the *error of the best piecewise-linear approximation to the data points* P as a recurrence relation among a set of sub-problems that you define. State the recurrence relation and its base cases and explain clearly why your recurrence allows you to correctly solve the problem. Your recurrence relation should allow you to solve the problem in $O(Kn^2)$ via dynamic programming.

Review: You should also be able to give a complete description of an $O(Kn^2)$ time dynamic programming algorithm that takes a set of n points P as input, and outputs the best piecewise-linear approximation to the data points (the function itself, not just its error).

Network Flows and Cuts

Problem 2 (Short-Answer Questions, 20 points)**.**

(a) Either prove or give a counterexample to the following statement: Let $G = (V, E, s, t, \{c(e)\})$ be a flow network and let (A, B) be a minimum (s, t)-cut in G. Suppose that we increase the capacity of every edge by 1 to get a new flow network $G' = (V, E, s, t, \{c'(e) = c(e) + 1\})$. Then (A, B) is also a minimum (s, t)-cut in G'.

(b) Now we will answer the question plaguing the military strategists of Qatar[1]. Let

$$G = (V, E, s, t, \{c(e) = 1\}_{e \in E})$$

be a flow network where every edge has capacity 1 and let $v^*(G)$ be the value of the maximum (s, t)-flow in G. Suppose we remove some set of at most k edges from G to obtain a new graph G', and let $v^*(G')$ be the value of the maximum (s, t)-flow in G. Our goal is to choose the edges to remove so that $v^*(G')$ is as small as possible. Consider the following algorithm:

> Find any minimum (s, t)-cut (A, B) in G and remove any set of k edges that cross from A to B. If there are only $k' < k$ such edges, then remove all k' of them.

Either prove that this algorithm makes $v^*(G')$ as small as possible or give a counterexample.

Review: Setting aside the issue of whether or not it actually solves the problem, show how to implement this algorithm in time $O(mn)$.

[1] Formerly, the United Cutdom of Great Britain and Northern Ireland.

Problem 3 (Cellular Networks, 30 points)**.**

Consider a set of cell phones that need to be connected to one of several base stations. We'll suppose there are n phones, and each phone i is specified by its location (x_i, y_i) in the plane. There are k base stations, and each base station j is specified by its location (x_j, y_j) in the plane. Every phone i must be connected to exactly one base station, but has a limited range r_i, and cannot be connected to any base station at a distance greater than r_i. On the other hand, each base station j has a load parameter L_j and cannot provide service to more than L_j phones at a time—so we cannot necessarily just connect every phone to its nearest base station.

Suppose you have access to a subroutine MaxFlow that computes the maximum flow in a given network. Use this subroutine to design a polynomial-time algorithm that takes a set of positions, ranges, and loads for n phones and k base stations and either 1) outputs a way to connect the phones to the base stations while satisfying the range and load constraints or 2) says that it is not possible to connect all n phones. Prove correctness and analyze the running time. Your running time analysis should include the time required by MaxFlow when implemented with one of the algorithm we've seen in class.[2]

Problem 4 (Flow with Edge Demands, 35 points)**.**

Suppose that instead of capacities, we consider networks where each edge $u \to v$ has some non-negative, integer-valued *demand* $d(u \to v)$. Now, we say that an (s,t)-flow is *feasible* if $f(u \to v) \geq d(u \to v)$ fo every edge $u \to v$, in addition to satisfying non-negativity and flow-conservation. Note that there are no capacities at all, so feasible flows can be arbitrarily large. A natural problem in this setting is to find a feasible flow of *minimum* value.

(a) Describe an efficient algorithm to compute *some* feasible (s,t)-flow, not necessarily the feasible flow of minimum value.[3] Prove that your algorithm finds a feasible flow and analyze its running time.

(b) Suppose you have access to a subroutine MaxFlow that computes maximum flows in networks with edge capacities (i.e. the standard maximum flow problem we've been studying). Describe an efficient algorithm to compute a minimum flow in a network with edge demands; your algorithm should call MaxFlow exactly once.[4] Prove that your algorithm is correct and analyze its running time. Your running time analysis should include the time required by the subroutine MaxFlow when implemented using one of the algorithms we've seen in class.

(c) State and prove an analogue of the max-flow/min-cut theorem for this setting. Do minimum flows correspond to maximum cuts?

[2] But, you may assume that algebraic operations (e.g. addition, multiplication, comparison, etc.) on the numbers x, y, r, L in the input takes $O(1)$ time.

[3] Hint: Start by finding a flow that is non-zero everywhere.

[4] Hint: Start with the feasible (s,t)-flow your algorithm finds in part (a) and try to *remove* as much flow as possible while still satisfying the demands.