CS7800: Advanced Algorithms. Fall 2017
Homework 5

Instructor: Jonathan Ullman TA: Albert Cheu

Due Friday, October 13th at 11:59pm
(Email to neu.cs7800@gmail.com)

Homework Guidelines

Collaboration Policy. Collaboration on homework problems is permitted, however it will serve you well on the exams if you solve the problems by yourself. If you choose to collaborate, you may discuss the homework with at most 2 other students currently enrolled in the class. Finding answers to problems on the Web or from other outside sources (these include anyone not enrolled in the class) is strictly forbidden.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You must also identify your collaborators. If you did not work with anyone, indicate that on your submission. If asked, you must be able to explain your solution to the instructors.

Preparing and Submitting Solutions. You must type your solutions using LaTeX. Please use an 11-pt or larger font. Please submit both the source and PDF files using the naming conventions `lastname_hw5.tex` and `lastname_hw5.pdf`. Your name must be on the first page of the PDF.

Solution guidelines For problems that require you to provide an algorithm, you must give the following: (1) a precise description of the algorithm in English and, if helpful, pseudocode, (2) a proof of correctness, (3) an analysis of running time. You may use any facts from class in your analysis and you may use any algorithms from class as subroutines in your solution.

You should be as clear and concise as possible in your write-up of solutions. Communication of technical material is an important skill, so clarity is as important as correctness. A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand. Points might be subtracted for solutions that are too long.

Note: I apologize for my inability to come up with funny stories to attach to this week's question.

Problem 1 (Number of Shortest Paths, 10 pts)**.**

In class, we saw an efficient dynamic programming algorithm (Bellman-Ford) that takes as input a weighted graph $G = (V, E, \{w_e\})$ and a pair of nodes $s, t \in V$ and outputs a shortest path from s to t. You may assume that any cycle in the graph has strictly positive weight. Design a polynomial time algorithm that takes the same input and returns the *number* of distinct shortest paths from s to t. Clearly state your algorithm, prove that it is correct, and analyze its running time and space using.

Problem 2 (Critical Edges in Flow Networks, 14 pts)**.**

Let $G = (V, E, \{c_e\})$ be a network with integral edge capacities. We say that an edge e is *flow-enchancing* if increasing its capacity by 1 also increases the value of the maximum flow in G. Similarly, an edge s is *flow-reducing* if decreasing its capacity by 1 also decreases the value of the maximum flow in G.

In this problem you will design algorithms for finding all flow-enhancing and flow-reducing edges in a graph. A trivial algorithm to determine if a specific edge is flow-enhancing or flow-reducing is to compute the maximum flow in G, then change the capacity of the edge and recompute the maximum flow to see if its value changes. Computing a maximum flow using the algorithms you've seen takes at least $O(mn)$ time, so the overall time to check all edges is $O(m^2 n)$. In this problem you will see that there are much faster and simpler algorithms.

(a) Does every network G have at least one flow-enhancing edge? Either prove that the answer is yes or give a counterexample.

(b) Describe and analyze an algorithm to find all flow-enhancing edges in G, given both G and a maximum flow in G as input. Your algorithm should run in at most $O(m^2)$ time and faster algorithms will score higher.

(c) Does every network G with a non-zero maximum flow have at least one flow-reducing edge? Either prove that the answer is yes or give a counterexample.

(d) Describe and analyze an algorithm to find all flow-reducing edges in G, given both G and a maximum flow in G as input. Your algorithm should run in at most $O(m^2)$ time and faster algorithms will score higher.