CSE 521: Design and Analysis of Algorithms — Winter 2006

Take Home Midterm — Instructor: Venkatesan Guruswami

Due on **February 14, 2006** in class.

Instructions: This is a take home exam with the following rules and instructions:

- The work you turn in should be entirely your own. You are not allowed to collaborate or discuss the problems, solutions, or any aspect relating to this exam with your classmates, or anyone else. If you have some difficulty following any of the questions or think something is ambiguous, send an email to the course instructor/TA.
- You are ONLY permitted to refer to the Kleinberg-Tardos text, your class notes and solutions to previous problem sets. Reference to any other source will be considered an act of academic dishonesty.
- Devote sufficient time for writing your solutions in a clear manner. It will help our evaluation, and also help you be more convinced of the correctness of what you turn in.
- If for some reason you cannot make it to class on Tuesday, make **prior** arrangements with us to turn in your exam before class on Tuesday.
- Attempt all questions. The exam is on 60 points. I think the exam looks longer than it really is, but if the length causes you to panic a bit, let me reassure you by saying that solving 4 out of the 5 problems correctly will get you a good (about 75% on an absolute scale) score.

---

1. ($5+5=10$ points) State whether the following statements are True of False, and justify your answer.

   (a) Let $G=(V,E)$ be a directed graph. Let $a,b,c \in V$ be three distinct vertices such that in the graph G there exist k mutually edge-disjoint paths from a to b, as well as k mutually edge-disjoint paths from b to c. Then there also exist k mutually edge-disjoint paths between a and c.

   (b) Consider the following "Forward-Edge Only" algorithm for computing s-t flows. The algorithm runs in a sequence of augmentation steps till there is no s-t path in the residual graph, except that we use a variant of the residual graph that *only includes the forward edges*. In other words, the algorithm searches for s-t paths in a graph $\tilde{G}_f$ consisting only of edges e for which $f(e) < c(e)$, and terminates when there is no augmenting path consisting entirely of such edges. Note that we do not prescribe how this algorithm chooses its forward-edge paths, it may choose them in any fashion it wants, provided that it terminates only when there are no forward-edge paths.

   Now to our claim: On every instance of the Maximum Flow problem, the forward-edge only algorithm returns a flow with value at least 1/4 of the maximum-flow value (regardless of how it chooses it forward-edge paths).

2. (12 points) Given a sequence σ of distinct integers $x_1, x_2, \ldots, x_n$, a *monotonic subsequence* of σ of length $\ell \geq 1$ is a sequence $x_{i_1}, x_{i_2}, \ldots, x_{i_\ell}$ such that $1 \leq i_1 < i_2 < \cdots < i_\ell \leq n$ and either $x_{i_1} < x_{i_2} < \cdots < x_{i_\ell}$ or $x_{i_1} > x_{i_2} > \cdots > x_{i_\ell}$. Give an algorithm that on input a sequence of distinct integers, finds a monotonic subsequence of maximum length. For full credit, your algorithm should use only $O(n \log n)$ comparisons where n is the length of the input sequence.
Suggestion: First come up with a polynomial time algorithm using a dynamic programming like approach — a natural way to create subproblems is to consider prefixes of the input sequence, but you will have to figure what exactly to store about the prefixes. The polytime algorithm should get you a good amount of the credit; and then worry about optimizing the runtime.

3. ($6 + 6 = 12$ points) This problem concerns Boolean circuits for computing the majority of n bits. Formally, the majority function $\mathsf{Majority}_n$ on n bits is defined by:

$$\mathsf{Majority}_n(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i \geq n/2 \\ 0 & \text{otherwise} \end{cases} .$$

We wish to find small circuits to compute $\mathsf{Majority}_n$. In our circuits, we allow three kinds of gates: AND, OR, and NOT. We restrict the fan-in of the AND and OR gates to two (i.e., these gates compute the AND and OR of their two input bits), but they can have arbitrarily large fan-out (i.e., their output can be connected as input to as many other gates as we choose). The NOT gates negate their input bit, and can also have any fan-out. Mathematically, the gates and connecting wires form a directed acyclic graph (DAG). The size of a circuit is the number of gates in it.

   (a) Give an algorithm, that on input n, runs in time polynomial in n and outputs a circuit of size $O(n)$ to compute $\mathsf{Majority}_n$. (Hint: Divide and conquer.)

   (b) Note that $\mathsf{Majority}_n$ is a *monotone* function, i.e., flipping an input bit from 0 to 1 never causes $\mathsf{Majority}_n$ to change value from 1 to 0. Every monotone function admits a circuit using only AND and OR gates that computes it (convince yourself why this is true) – such a circuit, that does not use any NOT gates, is called a *monotone circuit.*

   Give an algorithm, that on input n, runs in time polynomial in n and outputs a *monotone circuit* of size $O(n^2)$ to compute $\mathsf{Majority}_n$.

4. (10 points) Let $G = (V, E)$ be a directed graph, with source $s \in V$ and sink $t \in V$, and nonnegative edge capacities. Give a polynomial-time algorithm to decide whether G has a *unique* minimum s-t cut, or in other words whether G has an s-t cut of capacity *strictly less* than that of *all other* s-t cuts.
Hint: Consider the residual graph at the termination of the Ford-Fulkerson algorithm, and define minimum s-t cuts based on this graph in two natural ways. When are these two cuts actually the same cut?

5. (16 points) Consider the following variant of the minimum cut problem, where instead of two nodes s and t, we are given an undirected graph with nonnegative edge capacities along with *three* distinct vertices s_1, s_2, s_3 called terminals. The goal is to find a cut (i.e., a set of edges) whose removal separates each s_i from the other two terminals, i.e., places each of s_1, s_2, s_3 in a different connected component. Let us call such a cut a 3-way cut. Analogous to the

minimum cut problem, we have the problem of computing a 3-way cut of minimum capacity. This problem studies a natural greedy approach to this problem.

(a) (3 points) Given an undirected graph with nonnegative edge capacities and terminals s_1, s_2, s_3, define an isolating cut for s_i to be a cut that separates s_i from the other two terminals (but it need *not* separate those other two terminals). Give a polynomial time algorithm that finds a minimum capacity isolating cut for s_i, for $i = 1, 2, 3$.

(b) Now consider the following greedy heuristic for finding a 3-way cut.

- For each $i = 1, 2, 3$, find a minimum cost isolating cut, say C_i, for s_i.
- Let C_{i_1} and C_{i_2} be the two cheapest cuts among C_1, C_2, C_3 in terms of their capacity.
- Output $C_{i_1} \cup C_{i_2}$.

Now to your questions concerning this algorithm:

i. (2 points) Argue that the above algorithm indeed outputs a 3-way cut.

ii. (3 points) Is the following claim true: The above algorithm always returns a 3-way cut of minimum capacity? Justify your answer.

iii. (8 points) What is the smallest value of the absolute constant $\alpha \geq 1$ for which the following statement is true? : On every 3-way cut instance, the above algorithm always returns a 3-way cut whose capacity is at most α times the capacity of a minimum 3-way cut. Prove your answer.