EECS 495, Winter 2010
Instructor: Nicole Immorlica
Problem Set #1
Due: Feb 1, 2010

1. **Generating random bits.**

   Consider the following scenario: you buy a coin with bias p, then an adversary asks you to simulate a coin with bias q. The goal of the puzzle is to determine the original bias p that will minimize the expected number of coin flips required to simulate the adversarily chosen q (note that q is a function of p). I do not know how to do this (in fact, it or close variants are likely open questions and hence a good research project for this course). Instead this problem focuses on using a coin of unknown bias to produce a coin of bias 1/2.

   (a) It is natural to assume that our sources of randomness might have some bias p. Nonetheless, many randomized algoritms require unbiased coin flips. In 1951, von Neumann suggested the following procedure for producing an unbiased coin flip from a biased coin: Flip the biased coin twice. If the observed sequence is HT output H; if it is TH output T; else start over. Calculate the expected number of coin flips that this procedure uses to create one unbiased coin flip.

   (b) Note that von Neumann's method works even if you do not know the bias p. Now suppose $p = 2/3$. Design a procedure that improves upon von Neumann's method and calculate the expected number of flips required.

   (c) Again supposing you don't know p, design a procedure that provably improves upon von Neumann's method. It is great if you can analyze your procedure, but it's sufficient for the purposes of the problem set to prove that it beats von Neumann.

2. Understanding duality.

Consider a directed[1] graph G with source s and sink t, and positive arc capacities $c : A \rightarrow \Re^+$.

The problem of computing a max $s{-}t$ flow in a graph is: find the maximum flow that can be sent from s to t subject to the following constraints:

- *capacity constraints*: for each arc e, the flow sent through e is at most its capacity c_e, and
- *flow conservation*: at each node $v \neq s, t$, the total flow into v equals the total flow out of v.

For a partition of the nodes into two sets X_1 and X_2 with $s \in X_1$, the capacity of the cut $c(X_1, X_2)$ is the sum of the capacities of the arcs from X_1 to X_2. The min $s - t$ cut is the cut with smallest capacity.

This problem asks you to investigate the relationship between max flows and min cuts using LP duality.

(a) Formulate the max $s - t$ flow problem as a linear program.[2]

(b) Write the dual of your LP.

(c) Interpret the integral version of your dual LP as the min $s - t$ cut, concluding that the max $s{-}t$ flow lower bounds the min $s{-}t$ cut. Be sure to explain your reasoning, including how the integral solutions map to cuts and why this implies that the max $s{-}t$ flow lower bounds the min $s - t$ cut.

(d) Prove that the dual LP is in fact integral (i.e., the polytopes of the vertex are integral), and thus conclude that the max $s - t$ flow in fact *equals* the min $s - t$ cut. It may be useful to read up on total unimodularity (although I'm not sure this is necessary).

[1] This is WLOG since if we are given an undirected graph we can replace each edge with two arcs of the same capacity, one in each direction.

[2] HINT: First introduce a fictituous arc from t to s so that we are dealing with a circulation. Now label each arc (i, j) with a variable f_{ij} representing the flow on arc (i, j).

3. Designing simple algorithms.

Given a graph G on n vertices with m edges and an integer k, the long-path problem asks us to find a simple path of length at least k in graph G. For $k = n - 1$ this is the Hamiltonian path problem, which is NP-hard. In this problem, we will design a polytime randomized algorithm due to Alon, Yuster, and Zwick for $k = O(\log n)$.

(a) First show how to use dynamic programming to find the longest path in a directed acyclic graph (DAG) which runs in time linear in the number of edges.

(b) Consider ordering the vertices of G randomly from left to right and direct all edges forward. This creates a DAG to which we can apply the procedure from the previous part. Prove that if G had a simple path of length k, then the resulting DAG has a directed path of length at least k with probability at least $2/(k+1)!$.

(c) Use these two observations to design a polytime Monte Carlo algorithm which finds a path of length $k = O(\log n / \log\log n)$ with high probability if one exists.

(d) To make this work for longer paths, consider the following technique: color each vertex randomly with one of $k+1$ colors. Call a path colorful if no color repeats (note the longest colorful path can have length k). First show how to use dynamic programming to find a long colorful path in time $O(2^k mk)$. Then use Stirling's formula to bound the probability that a given long path is colorful by a reverse-exponential in k. Conclude that there's a polytime algorithm that finds a path of length $O(\log n)$ with high probability if one exists.

4. **Minimax principle.**

The purpose of this problem is to prove Yao's minimax method through LP duality and practice an application of it to sorting. Given an $n \times m$ payoff matrix A for a zero-sum game, let Δ_n be the set of mixed strategies for the row player and Δ_m be the set of mixed strategies for the column player. We want to prove von Neumann's minimax theorem,

$$\min_{x \in \Delta_m} \max_{y \in \Delta_n} y^T A x = \max_{y \in \Delta_n} \min_{x \in \Delta_m} y^T A x.$$

(a) First express the problem of computing a best-response as an LP and show that the LP is integral (and hence we can assume there is a best-response that is a pure strategy). Hence we have reduced the problem to proving

$$\min_{x \in \Delta_m} \max_{i \in \{1,n\}} A_i x = \max_{y \in \Delta_n} \min_{j \in \{1,m\}} y^T A_j.$$

(b) Write the lhs of the statement in the previous part as an LP, take the dual, and show it is equal to the rhs of the statement.[3] Conclude von Neumann's minimax theorem via LP duality.

(c) Use Yao's minimax principle to show that the randomized quicksort algorithm presented in lecture is optimal.

[3]HINT: Often times you will see objectives that don't look quite linear, like in this problem where we have the min of a max. To deal with such objectives, add a variable representing the max and a constraint forcing the max to be at most this variable and then minimize the variable in the objective function. Such tricks also help you deal with max of min objectives or objectives maximizing or minimizing a ratio.