CSE 521: Design and Analysis of Algorithms — Winter 2006
Problem Set #5 — Instructor: Venkatesan Guruswami
Due on **March 9, 2006** (Thursday) in class.

Instructions: Same as for Problem Set 1.

Readings: Kleinberg and Tardos, Sections 11.2, 11.8; Sections 13.1-13.4, 13.6. Read Sections 13.12 and 13.9 for background material on probability.

1. (12 points) Consider the following variant of the set cover problem. We are given a universe U of n elements and a collection $\mathcal{F} = \{S_1, S_2, \dots, S_m\}$ of subsets of U. The goal is to pick a subfamily $\mathcal{G}$ of $\mathcal{F}$ to maximize the number of elements of U which are covered *exactly once* by this subfamily.

   (a) Suppose each element of U is present in exactly k sets. Give a randomized algorithm that outputs a subfamily which uniquely covers a number of elements which is in expectation at least $1/e$ times the optimal value.

   How does your analysis change if each element u is contained in k_u of the sets, where $k \le k_u \le 2k$ for all $u \in U$?

   (b) Using the above algorithm and classifying elements into suitable groups, obtain a $O(\log B)$ approximation algorithm for the general problem, where B is the maximum number of sets to which any element of U belongs.

2. (12 points) In this problem, we will revisit the Contraction algorithm for computing minimum cuts, and consider its ability to find near-minimum cuts. For an integer $\ell \ge 1$, define an ℓ-approximate cut to be a cut whose size is at most ℓ times the size of the minimum cut. (We are considering unweighted, undirected graphs in this problem.)

   (a) Prove that a single trial of the contraction algorithm yields as output an ℓ-approximate cut with probability at least $\Omega(n^{-2/\ell})$, where n is the number of vertices in the graph.

   (b) For each fixed integer $\ell \ge 1$, give a polynomial time algorithm that outputs a list of all ℓ-approximate cuts in the graph. Prove also that, in any n-vertex graph, there are at most $n^{2\ell}$ ℓ-approximate cuts.

3. (10 points) Consider the problem of finding a subset S of vertices of an unweighted, undirected graph $G = (V, E)$ that maximizes the density $\rho(S) = \frac{|E(S)|}{|S|}$ where $E(S)$ is the set of edges both of whose endpoints belong to S. In Problem Set 3, you were asked to give a flow-based polynomial time algorithm for this problem. In this exercise, you will prove that a simpler algorithm delivers a good approximation. The problem is equivalent to finding a subgraph of G of largest average degree. Intuitively, we should throw away low-degree vertices to produce such a subgraph. This motivates the following natural greedy approach.

   > The algorithm maintain a subset S of vertices. Initially $S = V$. In each iteration, the algorithm finds $v_{\min}$, the vertex of minimum degree in the subgraph $G[S]$ induced by S. It then removes $v_{\min}$ from S and move on to the next iteration. The process terminates when the set S is empty. Of all the sets S constructed during the various iterations of the algorithm, the algorithm returns the set S maximizing $\rho(S)$ as the output.

Prove that the above is a 2-approximation algorithm for the problem of computing a set with largest density $\rho(S)$.

4. (16 points) For this problem you can use the following fact concerning polynomials.

   Let p be a prime and $\mathbb{F}_p = \{0, 1, \ldots, p-1\}$. Let $Q(X_1, X_2, \ldots, X_m)$ be a **nonzero** polynomial in m variables with coefficients being integers from $\mathbb{F}_p$, with the degree of Q in each x_i being at most d. Then, if r_i is picked uniformly at random from $\mathbb{F}_p$ for each i independently, the probability (over the choice of the $r_1, \ldots, r_m$) that $Q(r_1, r_2, \ldots, r_m) \equiv 0 \pmod{p}$ is at most md/p.

   You can also use the fact there is always a prime between M and $2M$ for $M \geq 2$.

   (a) Suppose Alice has an n-bit string $a \in \{0,1\}^n$ and Bob has an n-bit string $b \in \{0,1\}^n$. Alice wishes to send a single message to Bob upon receiving which Bob can ascertain whether $a = b$ or not. Of course one obvious way for Alice to achieve this goal will be to send the entire string a itself. But this requires communicating n bits and Alice prefers to be lazy and transmit fewer bits if possible.

   i. Prove that every deterministic strategy for Alice that always lets Bob conclude the correct answer requires Alice to send n bits.

   ii. Demonstrate a randomized strategy for Alice and Bob under which Alice sends only $O(\log n)$ bits and Bob errs with probability at most $1/n$ in ascertaining whether or not $a = b$.

   (b) In this problem we revisit the question of existence of perfect matchings in bipartite graphs. We have already seen how to solve this question with a deterministic algorithm. In this exercise, you will develop a randomized algorithm for this task; this algorithm is perhaps somewhat simpler, and also (something we won't get into) is amenable to parallelization.

   i. Given a bipartite graph $H = (V, W, E)$ where $V = \{v_1, \ldots, v_n\}$ and $W = \{w_1, \ldots, w_n\}$, define an $n \times n$ matrix $A(H)$ as follows. For $1 \leq i, j \leq n$, the (i,j)'th entry of $A(H)$ is defined as

   $$A(H)_{i,j} = \begin{cases} x_{i,j} & \text{if } (v_i, w_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

   where each $x_{i,j}$ used is a new indeterminate. Prove that the determinant of $A(H)$, $\mathsf{det}(A(H))$, is nonzero as a polynomial in the x_{ij}'s if and only if H has a perfect matching.

   (Recall that for a matrix $B = \{b_{i,j}\}_{1 \leq i,j \leq n}$,

   $$\mathsf{det}(B) = \sum_{\sigma \in S_n} \mathsf{sgn}(\sigma) \prod_{i=1}^{n} b_{i,\sigma(i)}$$

   where the summation is over all permutations on $\{1, 2, \ldots, n\}$ and $\mathsf{sgn}(\sigma)$ is the "sign" of the permutation σ, where $\mathsf{sgn}(\sigma) = 1$ for even permutations and -1 for odd permutations.)

   ii. Use part (a) to give a randomized algorithm for perfect matching with the following properties: (i) if H has no perfect matching, the algorithm always rejects, and (ii) if H has a perfect matching the algorithm accepts with probability $1 - 1/n$.