CS 174: Combinatorics and Discrete Probability Fall 2012

Homework 4

Due: Thursday, September 27, 2012 by **9:30am**

Instructions: *You should upload your homework solutions on bspace. You are strongly encouraged to type out your solutions using LaTeX. You may also want to consider using mathematical mode typing in some office suite if you are not familiar with LaTeX. If you must handwrite your homeworks, please write clearly and legibly. We will not grade homeworks that are unreadable. You are encouraged to work in groups of 2-4, but you* **must** *write solutions on your own. Please review the homework policy carefully on the class homepage.*

Note: You *must* justify all your answers. In particular, you will get no credit if you simply write the final answer without any explanation.

Problem 1. *(Exercise 3.24 from MU – 5 points)* Recall the randomized algorithm discussed in class for finding the median of a set S of n elements (see MU, Section 3.4). Explain how to generalize this algorithm so that it takes an additional parameter k as input and finds the element of rank k in S (*i.e.* the k^{th} smallest element of S). For convenience, you may assume that all elements of S are distinct, and that $4n^{3/4} \leq k \leq n - 4n^{3/4}$.You may also ignore rounding issues in your algorithm.

Your answer should be *fairly short* and you should *not* repeat unnecessary material from the text. The only things you need to include are the following:

(i) a *specification* of the algorithm (in similar style to that in class and in the textbook); follow the same first step (namely, pick R of size $n^{3/4}$); highlight those points where your algorithm differs from the median case;

(ii) a *very brief outline* of the analysis of the algorithm, following the same path as in class and in the textbook; specifically, state the analogs of the events $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_{3,1}$ and $\mathcal{E}_{3,2}$, and express $\Pr[\mathcal{E}_1]$ and $\Pr[\mathcal{E}_{3,1}]$ as probability expressions involving the tail of a suitable binomial random variable. Do *not* repeat the explanation of the algorithm of the details of the calculations (which are essentially the same as in the median case); the above points are enough.

Problem 2. *(Exercise 4.10 from MU – 5 points)* A casino is testing a new class of simple slot machines. Each game, the player puts in \$1, and the slot machine is supposed to return either \$3 to the player with probability 4/25, \$100 with probability 1/200, or nothing with all the remaining probability. Each game is supposed to be independent of other games.

The casino has been surprised to find in testing that the machines have lost \$10,000 over the first million games. Your task is to come up with an upper bound on the probability of this event, assuming that their machines are working as specified.

(i) Let the random variable X denote the *net loss* to the casino over the first million games. By writing $X = X_1 + \cdots + X_{10^6}$, derive an expression for $\mathbb{E}[e^{tX}]$, where t is an arbitrary real number.

(ii) Derive from first principles a Chernoff bound for the probability $\Pr[X \geq 10,000]$. (You should follow the proof of the Chernoff bound in class, by applying Markov's inequality to the random variable e^{tX}. You should use the value $t = 0.0006$ in your bound.)

Problem 3 *(Exercise 5.4 from MU – 5 points)* In a lecture hall containing 100 people, you consider whether or not there are three people in the room who share the same birthday. Explain how to calculate this probability exactly. (You should make the following assumptions: None of the people is born in a leap year, a person is equally likely to be born on any day of the year, and that the birthdays of different people are independent of each other. No twins!)

Problem 4. (15 points) A fundamental problem that arises in many applications is to compute the size of the *union* of a collection of sets. The setting is the following. We are given m sets $S_1, \ldots, S_m$ over a very large universe U. The operations we can perform on the sets are the following:

(a) size(S_i): returns the number of elements in S_i;

(b) select(S_i): returns an element of S_i chosen uniformly at random;

(c) lowest(x): for some $x \in U$, returns the smallest index i for which $x \in S_i$.

Let $S = \cup_{i=1}^{m} S_i$ be the union of the sets S_i. In this problem we will develop a very efficient (polynomial in m) algorithm for estimating the size of $|S|$. (We output a number in the range $[(1-\epsilon)|S|, (1+\epsilon)|S|]$.)

1. Let's first see a natural example where such a set system arises. Suppose ϕ is a boolean formula over n variables in *disjunctive normal form* (DNF), *i.e.* it is the OR of ANDs of literals. ($\phi = C_1 \vee C_2 \vee \cdots \vee C_m$, where each C_i is a conjunction (AND) of possibly negated literals.) Let U be the set of all possible assignments to the variables of ϕ (*i.e.* $|U| = 2^n$), and for each clause $1 \leq i \leq m$, let S_i denote the set of assignments that satisfy the clause C_i. Then the union $S = \cup_{i=1}^{m} S_i$ is exactly the set of satisfying assignments of ϕ, and our problem is to count them.[1] Argue that all of the above operations can be efficiently implemented for this set system.

2. Now let's consider a naïve random sampling algorithm. Assume that we are able to pick an element of the universe, U, uniformly at random, and that we know the size of U. Consider an algorithm that picks t elements of U independently and u.a.r. (with replacement), and outputs the value $q|U|$, where q is the proportion of the t sampled elements that belong to S. For the DNF example above, explain as precisely as you can why this is not a good algorithm.

3. Consider now the following algorithm, which is again based on random sampling but in a more sophisticated way:

   - choose a random set S_i with probability $\frac{\text{size}(S_i)}{\sum_{j=1}^{m} \text{size}(S_j)}$.
   - $x = \text{select}(S_i)$
   - if lowest$(x) = i$, then output 1, else output 0

[1] Deciding if ϕ is satisfiable (*i.e.* has at least one satisfying assignment) is trivial for a DNF formula, unlike for a CNF formula where it is NP-complete. However, when it comes to *counting* satisfying assignments, it turns out that the problem is NP-hard even for DNF formulas! Thus, we cannot hope to find a polynomial time algorithm that solves this problem exactly. Thus, the approximation algorithm that we develop in this question is essentially the best one can hope for.

Show that this algorithm outputs 1 with probability exactly $p = \frac{|S|}{\sum_{j=1}^m |S_j|}$. (Hint: Show that the effect of the first two lines of the algorithm is to select a random element from the set of pairs $\{(x, S_i) \mid x \in S_i\}$.)

4. Show that $p \geq 1/m$.

5. Now suppose that we run the above algorithm t times and obtain the sequence of outputs $X_1, \ldots, X_t$. We define $X = \sum_{i=1}^t X_i$. Use a Chernoff bound to obtain a value t (as a function of m, δ and ϵ) that ensures that

$$\Pr[|X - tp| \geq \epsilon tp] \leq \delta$$

(Hint: You will need to use the fact tht $p \geq 1/m$ here.)

6. The final output of your algorithm will be $Y = (\sum_{j=1}^m |S_j|)\cdot(X/t)$, where X is as defined above. Show that this final algorithm has the following properties: it runs in time $O(m\epsilon^{-2}\log(1/\delta))$ (assuming that each of the set operations can be performed in constant time), and outputs a value that is in the range $[(1-\epsilon)|S|, (1+\epsilon)|S|]$ with probability at least $1-\delta$.