CS 174: Combinatorics and Discrete Probability Fall 2012

Homework 4 Solutions

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.

Solution: The algorithm is essentially the same as Algorithm 3.1 in the textbook:

1. *same as before*: Pick a set R of $n^{3/4}$ elements independently and u.a.r. from S
2. *same as before*: Sort R
3. Let d be the $(\frac{k}{n} \cdot n^{3/4} - \sqrt{n})$ smallest element in R
4. Let u be the $(\frac{k}{n} \cdot n^{3/4} + \sqrt{n})$ smallest element in R
5. *same as before*: By comparing every element of S to d and u, compute C, l_d, l_u
6. If $l_d > k$ or $l_u > n - k$ then FAIL
7. *same as before*: If $|C| \leq 4n^{3/4}$ then sort C else FAIL
8. Output the $(k - l_d)$th element in C

As for the median algorithm in the book, this algorithm always runs in linear time and either outputs the correct value or FAILs. For the analysis, we just need to bound the failure probability. To do this, we will make repeated use of the following lemma:

Let Y be a binomial r.v. with parameters $n^{3/4}$ and p. Then, $\mathbb{E}[Y] = pn^{3/4}$ and $\text{Var}[Y] = n^{3/4}p(1-p) \leq \frac{1}{4}n^{3/4}$. So, by Chebyshev's inequality, $\Pr[|Y - E[Y]| \geq pn] \leq \frac{1}{4}n^{-1/4}$.

Note that this same fact is used repeatedly in the analysis of the median algorithm in the textbook. We define m to be the element of rank k in S (so m is the desired output). We will consider the following three events, which are analogous to the events with the same names in the textbook:

$$\mathcal{E}_1 : Y_1 = |\{r \in R | r \leq m\}| \leq \frac{k}{n} \cdot n^{3/4} - \sqrt{n};$$
$$\mathcal{E}_2 : Y_2 = |\{r \in R | r \geq m\}| \leq (1 - \frac{k}{n}) \cdot n^{3/4} - \sqrt{n};$$
$$\mathcal{E}_3 : |C| > 4n^{3/4}.$$

Clearly the probability that the algorithm fails is at most $\Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] + \Pr[\mathcal{E}_3]$. We now bound each of these probabilities in turn:

– To analyze $\Pr[\mathcal{E}_1]$, we define a r.v. X_i indicating whether the i^{th} sample (of R) is $\leq m$. Then, $Y_1 = \sum_i X_i$ is a binomial r.v. with parameters $n^{3/4}$ and $\frac{k}{n}$. Therefore, by the above lemma,

$$\Pr[\mathcal{E}_1] = \Pr\left[Y_1 \leq \frac{k}{n} \cdot n^{3/4} - \sqrt{n}\right] < \frac{1}{4} n^{-1/4}$$

– To analyze $\Pr[\mathcal{E}_2]$, we define a r.v. X_i indicating whether the i^{th} sample (of R) is $\geq m$. Then, $Y_2 = \sum_i X_i$ is a binomial r.v. with parameters $n^{3/4}$ and $1 - \frac{k}{n}$. Therefore, again by the above lemma,

$$\Pr[\mathcal{E}_2] = \Pr\left[Y_2 \leq (1 - \frac{k}{n}) \cdot n^{3/4} - \sqrt{n}\right] < \frac{1}{4} n^{-1/4}$$

– To analyze $\Pr[\mathcal{E}_3]$, as in the book we consider two events: $\mathcal{E}_{3,1}$: at least $2n^{3/4}$ elements of C are greater than m; $\mathcal{E}_{3,2}$: at least $2n^{3/4}$ elements of C are smaller than m.

Clearly if $\mathcal{E}_3$ happens then so must at least one of $\mathcal{E}_{3,1}$ and $\mathcal{E}_{3,2}$, so $\Pr[\mathcal{E}_3] \leq \Pr[\mathcal{E}_{3,1}] + \Pr[\mathcal{E}_{3,2}]$. Let Γ_1 denote the $k - 2n^{3/4}$ smallest elements of S. $\mathcal{E}_{3,1}$ may be rewritten as: R contains $\frac{k}{n} \cdot n^{3/4} - \sqrt{n}$ elements of Γ_1. We let X be the number of samples (of R) in Γ_1. Then, $X = \sum_i X_i$ where X_i is a r.v. indicating whether the i^{th} sample lies in Γ_1. Again, X is a binomial r.v. with parameters $n^{3/4}$ and $\frac{k}{n} - 2n^{-1/4}$. Thus $\mathbb{E}[X] = \frac{k}{n} \cdot n^{3/4} - 2\sqrt{n}$. In addition,

$$\Pr[\mathcal{E}_{3,1}] = \Pr\left[X \geq \frac{k}{n} \cdot n^{3/4} - \sqrt{n}\right] < \frac{1}{4} n^{-1/4}$$

The analysis of $\Pr[\mathcal{E}_{3,2}]$ is symmetrical.

Putting all the above together, we see that the probability of failure is at most

$$\Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] + \Pr[\mathcal{E}_{3,1}] + \Pr[\mathcal{E}_{3,2}] \leq n^{-1/4}$$

as required.

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.

Solution: Let X_i, $i = 1, 2, \ldots, 10^6$ denote the casino's net loss in the i^{th} game. We have

$$X_i = \begin{cases} 2 & \text{w.p. } \frac{4}{25} \\ 99 & \text{w.p. } \frac{1}{200} \\ -1 & \text{w.p. } \frac{167}{200} \end{cases} \Rightarrow e^{tX_i} = \begin{cases} e^{2t} & \text{w.p. } \frac{4}{25} \\ e^{99t} & \text{w.p. } \frac{1}{200} \\ e^{-1t} & \text{w.p. } \frac{167}{200} \end{cases}$$

Therefore

$$\mathbb{E}[e^{tX_i}] = \frac{4}{25}e^{2t} + \frac{1}{200}e^{99t} + \frac{167}{200}e^{-1t}$$

Now, $X = X_1 + X_2 + \ldots + X_{10^6}$ is the casino's loss in the first million games, and we can compute $\mathbb{E}[e^{tX}]$ as follows:

$$\begin{aligned} \mathbb{E}[e^{tX}] &= E[e^{t(X_1+X_2+\ldots+X_{10^6})}] \\ &= \mathbb{E}[e^{tX_1}] \cdot \mathbb{E}[e^{tX_2}] \cdots \mathbb{E}[e^{tX_{10^6}}] \quad \text{since the } X_i \text{ are independent} \\ &= \left(\frac{4}{25}e^{2t} + \frac{1}{200}e^{99t} + \frac{167}{200}e^{-t}\right)^{10^6} \end{aligned}$$

(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.)

Solution: We are interested in the quantity

$$\begin{aligned} \Pr[X \geq 10^4] &= \Pr[e^{tX} \geq e^{10^4 t}] \\ &\leq \frac{E[e^{tX}]}{e^{10^4 t}} \quad \text{by Markovs inequality} \\ &= \left(\frac{4}{25}e^{2t} + \frac{1}{200}e^{99t} + \frac{167}{200}e^{-t}\right)^{10^6} e^{-10^4 t} \end{aligned}$$

This bound is valid for any $t > 0$, so we are free to choose a value of t that gives the best bound (i.e., the smallest value for the expression on the right). Plugging in $t = 0.0006$ as suggested in the hint, we get the bound 0.0002. This is very small, suggesting that the casino has a problem with its machines.

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!)

Solution: One possible solution is given here.

Let A_0 be the event that everyone in the room has a distinct birthday. Let A_i be the event that i pairs of people share a birthday. Then the probability we are interested in is

$$\Pr[\text{at least 3 people share a birthday}] = 1 - \Pr[A_0] - \sum_{i=1}^{n/2} \Pr[A_i]$$

where there are n people in the room.

First, we note that $\Pr[A_0]$ is in the book and is $\Pr[A_0] = \frac{\binom{365}{n} n!}{365^n}$. Second, we wish to compute $\Pr[A_i]$.

For $i = 1$, we have

$$\Pr[A_1] = 365 \frac{\binom{n}{2}\binom{365-1}{n-2}(n-2)!}{365^n}$$

For $i = 2$, we have

$$\Pr[A_2] = \frac{\binom{365}{2}\binom{n}{2}\binom{n-2}{2}\binom{365-2}{n-4}(n-4)!}{365^n}$$

This generalizes for i pairs of people sharing i distinct birthdays

$$\Pr[A_i] = \frac{\binom{365}{i}\prod_{j=1}^{i}\binom{n-2j+2}{2}\binom{365-i}{n-2i}(n-2i)!}{365^n}$$

Plugging everything into the first equation and putting in $n = 100$, we can compute a value for this probability. We get $\Pr[\text{at least 3 people share a birthday}] \approx 0.6459$.

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.

[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.

Solution: First, we process ϕ so that every variable appears at most once in each clause (eliminate repeated occurences of a literal, and delete a clause if both a literal and its negation occur). Let n denote the number of variables, and c_i the number of variables in clause i.

- size(x, S_i): return 2^{n-c_i}. The variables in clause i must be fixed to values that satisfy the clause, and the remaining variables may be assigned any value.
- select(S_i): fix the variables in clause i to values that satisfy the clause; choose the values of the remaining variables independently and u.a.r.
- lowest(x): for $i = 1, 2, \ldots$, test if x satisfies clause i (this test is easy); return the index of the first clause that x satisfies (or undefined if it satisfies no clauses).

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.

   Solution: The problem is that S may occupy only a tiny fraction of all possible assignments U. Thus the number of samples t would need to be huge in order to get a good estimate of q. We give a concrete example to make this precise. Consider the very simple formula $\phi = x_1 \wedge x_2 \wedge \ldots \wedge x_n$. Clearly $|S| = 1$ (the only satisfying assignment is when all n variables are TRUE). The given algorithm will output zero unless it happens to choose this assignment in one of its t samples, i.e., it outputs zero with probability $(12^n)^t \geq 1t2^n \approx 1$ for any t that is only polynomial in n. Thus the relative error of the algorithm will be arbitrarily large with probability arbitrarily close to 1.

   Note: It is not enough here to quote the bound from class $t = O(q/\epsilon^2 ln(1/\delta))$, which tells us how large a sample size is sufficient to estimate the proportion q. The reason is that this is an upper bound on t, whereas here we need a lower bound. The lower bound can be derived by the very simple argument given above.

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

   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 $\{(x, S_i) \mid x \in S_i\}$.)

   Solution: Note that the first two lines of the algorithm select each pair $(x, S_i), x \in S_i$ with probability $\frac{|S_i|}{\sum_{j=1}^{m} |S_j|} \cdot \frac{1}{|S_i|} = \frac{1}{\sum_{j=1}^{m} |S_j|}$. In other words, the first two lines pick an element u.a.r. from the disjoint union of the sets S_i. (Note that the goal is really to pick an element u.a.r. from the union $\bigcup_i S_i$.) Let $\Gamma = \{(x, S_i) | \text{lowest}(x) = i\}$. Clearly, the algorithm outputs 1 with probability $\sum_{(x,S_i)\in\Gamma} \frac{1}{\sum_{j=1}^{m} |S_j|} = \frac{|\Gamma|}{\sum_{j=1}^{m} |S_j|}$. To see that $|\Gamma| = |S|$, simply observe that every element $x \in S$ corresponds to exactly one lowest S_i, or equivalently $\Gamma = \{(x, S_{\text{lowest}(x)}) | x \in S\}$. It follows that the algorithm outputs 1 with probability $p = \frac{|S|}{\sum_{j=1}^{m} |S_j|}$.

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

   Solution: Clearly, for $i = 1, 2, \ldots, m$ we have $|S_i| \leq |S|$. Hence, $\sum_{j=1}^{m} |S_j| \leq m|S|$, and thus $p = \frac{|S|}{\sum_{j=1}^{m} |S_j|} \geq \frac{1}{m}$.

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.)

   Solution: Note that $X_1, \ldots, X_t$ are independent 0-1 r.v.'s with mean p, so $\mathbb{E}[X] = pt$ and the Chernoff bound yields

$$\Pr[|Xpt| \geq \epsilon pt] \leq 2e^{\epsilon^2 pt/3}$$

   The quantity on the right is bounded above by δ provided we take $t = \lceil \frac{3}{\epsilon^2 p} \ln \frac{2}{\delta} \rceil \leq \lceil \frac{3m}{\epsilon^2} \ln \frac{2}{\delta} \rceil$, using the fact from part (d) that $p \geq m$. Hence it suffices to take $t = O(\frac{m}{\epsilon^2} \log \frac{1}{\delta})$.

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 $1 - \delta$.

   Solution: Each iteration of the algorithm in (c) requires $O(1)$ operations, so the final algorithm takes $O(t) = O(\frac{m}{\epsilon^2} \log \frac{1}{\delta})$ time. By definition, we have $|S| = \frac{\sum_{j=1}^{m} |S_j|}{t} \cdot tp$ and $Y = \frac{\sum_{j=1}^{m} |S_j|}{t} \cdot X$. This implies

$$Y \in [(1-\epsilon)|S|, (1+\epsilon)|S|] \Leftrightarrow X \in [(1-\epsilon)tp, (1+\epsilon)tp]$$

   and thus

$$\Pr\left[Y \in [(1-\epsilon)|S|, (1+\epsilon)|S|]\right] = \Pr\left[X \in [(1-\epsilon)tp, (1+\epsilon)tp]\right]$$

   It follows by part (e) that $\Pr\left[Y \in [(1-\epsilon)|S|, (1+\epsilon)|S|]\right] \geq 1 - \delta$, as required.