COMS 4995 (Randomized Algorithms): Problem Set #3

Due by 11:59 PM on Wednesday, October 30, 2019

Instructions:

(1) Form a group of 1-3 students. You should turn in only one write-up for your entire group.

(2) Submission instructions: We are using Gradescope for the homework submissions. Go to www.gradescope.com to either login or create a new account. Use the course code 9D6V5E to register for this class. Only one group member needs to submit the assignment. When submitting, please remember to add all group member names in Gradescope. See the course Web site for detailed instructions.

(3) Please type your solutions if possible and we encourage you to use the LaTeX template provided on the course home page.

(4) Write convincingly but not excessively.

(5) Some of these problems are difficult, so your group may not solve them all to completion. In this case, you can write up what you've got (subject to (3), above): partial proofs, lemmas, high-level ideas, counterexamples, and so on.

(6) Except where otherwise noted, you may refer to your lecture notes and the specific supplementary readings listed on the course Web page *only.* You can also review any relevant materials from your undergraduate algorithms course. If you do use any approved sources, make you sure you cite them appropriately, and make sure that all your words are your own.

(7) You can discuss the problems verbally at a high level with other groups. And of course, you are encouraged to contact the course staff (via Piazza or office hours) for additional help.

(8) If you discuss solution approaches with anyone outside of your group, you must list their names on the front page of your write-up.

(9) Refer to the course Web page for the late day policy and the School of Engineering honor code.

Problem 13

(20 points) The point of this problem is to extend the Chernoff bound (for 0-1 random variables) to the Hoeffding bound (for random variables with bounded support).

Recall that a function $f : \mathbb{R} \to \mathbb{R}$ is *convex* if every chord between two points of the graph stays above the graph: for all $x, y \in \mathbb{R}$ and $\lambda \in [0,1]$,

$$\underbrace{f(\lambda x + (1-\lambda)y)}_{\text{corresponding point on } f\text{'s graph}} \quad \le \quad \underbrace{\lambda f(x) + (1-\lambda) f(y)}_{\text{point on chord between } (x,f(x)) \text{ and } (y,f(y))} \quad .$$

(If it's helpful, feel free to use the slightly stronger assumption that f is twice differentiable with $f''(x) \ge 0$ for all x.)

(a) (5 points) Prove *Jensen's inequality*: for every convex function f and random variable X (with finite expectation),

$$f(\mathbf{E}[X]) \le \mathbf{E}[f(X)] \, .$$

[Hint: consider a line tangent to the graph of f.]

(b) (7 points) Prove that the basic Chernoff bound (as stated in Exercise 24) holds more generally for sums $X = \sum_{i=1}^{n} X_i$ of independent random variables with support in $[0,1]$.[1]

[Hint: Use (a) to get an upper bound on the moment generating function $\mathbf{E}\big[e^{tX}\big]$ that matches the one we used in the proof in class.]

(c) (8 points) Now suppose that $X = \sum_{i=1}^{n} X_i$ where $X_1, \ldots, X_n$ are independent random variables with support in the interval $[a,b]$. State and prove a generalization of the Hoeffding bound in (b) for this case.

[Hint: first reduce to the case where $a = 0$.]

Problem 14

(12 points) Consider a symmetric random walk on the integers[2], meaning the following random process:

1. Initialize $X_0 = 0$.
2. For $t = 1, 2, \ldots,$
   (a) With 50% probability, $X_i = X_{i-1} + 1$.
   (b) Otherwise, $X_i = X_{i-1} - 1$.

Consider the following statement:

"For every positive integer T, with probability at least $1 - 1/T$, for all $i = 1, 2, \ldots, T$, $|X_i| = O(f(T))$."

Identify the smallest function (up to constant factors) $f(\cdot)$ that makes this statement true, and prove that your answer is correct (the proof should include upper and lower bounds that match up to a constant factor).

Problem 15

(15 points) In the *2SAT* problem, you are given a set of clauses, each of which is the disjunction (logical 'or") of two literals. (A literal is a Boolean variable or the negation of a Boolean variable. Assume that every clause involves two distinct variables.) You would like to assign a value 'true" or 'false" to each of the variables so that all the clauses are satisfied, with at least one true literal in each clause. For example, if the input contains the three clauses $x_1 \vee x_2$, $\neg x_1 \vee x_3$, and $\neg x_2 \vee \neg x_3$, then one way to satisfy all of them is to set x_1 and x_3 to 'true" and x_2 to 'false."[3] Of the seven other possible truth assignments, exactly one satisfies all three clauses.

Consider the following (Monte Carlo with 1-sided error) randomized algorithm for the 2SAT problem:

1. Begin with a random truth assignment, with each variable x_i set independently to "true" or "false" with 50/50 probability.
2. Repeat T times:
   (a) If the current truth assignment satisfies every clause, return "satisfiable."
   (b) Else, choose an arbitrary unsatisfied clause and randomly reassign both of its variables (again, independently to "true" or "false" with 50/50 probability).
3. Return "unsatisfiable."

How large does T need to be so that, for every input, this algorithm is correct with probability at least 2/3?

[Hint: for a satisfiable input, fix an arbitrary satisfying truth assignment and track the number of matching variable assignments in the current (random) assignment. Problem 14 is very relevant here.]

[1] That is, each X_i takes on some value in $[0,1]$ with probability 1.

[2] As performed by a sufficiently inebriated individual trying to find their way home.

[3] The symbol "$\vee$" stands for the logical "or" operation, while "$\neg$" denotes the negation of a Boolean variable.

Problem 16

(16 points) The goal of this problem is to prove that the Randomized QuickSort algorithms runs in $O(n \log n)$ time with high probability (rather than merely in expectation), where n is the length of the input array.

Consider the recursion tree of the algorithm.[4] Recall that each recursive call (other than the base cases) selects a pivot element uniformly at random from the subarray passed to the call. Call an internal node *good* if the pivot element chosen in the corresponding recursive call splits its input subarray (with length ℓ, say) into two subarrays with at most two-thirds the length (i.e., with at most $2\ell/3$ elements each). The other internal nodes are called *bad*.

(a) (4 points) Show that the number of good nodes in any path from the root to a leaf of the recursion tree is at most $c \log_2 n$, where c is some positive constant.

(b) (4 points) Show that, with high probability (at least $1 - 1/n^2$), the number of nodes in a given root-to-leaf path is at most $c' \log_2 n$, where c' is another constant.

(c) (4 points) Show that, with high probability (at least $1 - 1/n$), the number of nodes in the longest root-to-leaf is at most $c' \log_2 n$.

[Hint: how many nodes are in the tree in total?]

(d) (4 points) Deduce that the running time of Randomized QuickSort is $O(n \log n)$ with probability at least $1 - 1/n$.

Problem 17

(20 points) For the purposes of this problem, an undirected graph $G = (V, E)$ is an *expander* if for every subset $S \subseteq V$ of at most half the vertices, $|N(S)| \geq \frac{5}{4}|S|$. (Here $N(S)$ denotes the "neighbors" of S—the vertices of V that are adjacent to at least one vertex of S.) A simple example of an expander is a clique. But can sparse graphs also be expanders?

Consider the following randomized algorithm for generating a (sparse) graph. Start with a set V of an even number n of vertices. For $i = 1, 2, \ldots, d$, choose a perfect matching M_i of V (i.e., a partition of V into pairs) uniformly at random from all such matchings. (With different M_i's chosen independently of each other.) Take the union of the M_i's to form a graph G. Note that every vertex of G has degree at most d.

Prove that, provided d is at least a sufficiently large constant (independent of n), with high probability (at least $1 - 1/n$), the output of this randomized algorithm is an expander graph.

Problem 18

(17 points) This problem concerns graph sparsification, which is a useful preprocessing step to speed up certain graph algorithms.

Let $G = (V, E)$ be an undirected graph with n vertices. Recall that a *cut* of G is a partition (A, B) of its vertex set V into two non-empty sets. The *value* of a cut (A, B) is the number of edges with one endpoint in each of A and B. A $(1 + \epsilon)$-*cut sparsifier* for G is a graph $\widehat{G} = (V, \widehat{E})$ with the same vertex set and approximately the same cut values (up to a scaling factor): for some $\alpha > 0$,

$$\alpha \cdot \hat{v}(A, B) \in [(1 - \epsilon)v(A, B), (1 + \epsilon)v(A, B)]$$

for every cut (A, B). (Here $v(A, B)$ and $\hat{v}(A, B)$ denote the value of the cut (A, B) in G and $\widehat{G}$, respectively.)

We can obtain a (random) sparser graph $\widehat{G} = (V, \widehat{E})$ by independently including each edge $e \in E$ in $\widehat{E}$ with probability p, where $p \in [0, 1]$ is a parameter.

[4]Recall that every recursive algorithm can be associated with a recursion tree, in which the nodes of the tree correspond to all the algorithm's recursive calls. The root of the tree corresponds to the outermost initial call to the algorithm (with the original input), with one child at the next level for each of its recursive calls. The leaves at the bottom of the tree correspond to the recursive calls that trigger a base case and make no further recursive calls.

(a) (13 points) Prove that if every cut of G has value at least k and $p = \min\{\frac{c \ln n}{\epsilon^2 k}, 1\}$, where c is a sufficiently large constant (independent of n), then the output graph $\widehat{G}$ is a $(1+\epsilon)$-cut sparsifier of G with high probability (at least $1 - 1/n$, say).

[Hint: Bucket the terms in your union bound over cuts according to cut values in G. Use Problem 3 from Problem Set #1.]

(b) (4 points) Prove that if G is the complete graph and p is set as in (a), then with high probability the output graph has only $O(n \log n)$ edges.