PRINCETON UNIV FALL '20 COS 521:ADVANCED ALGORITHMS

Homework 1

Out: *Sep 1* Due: *Sep 18*

Instructions:

- Upload your solutions (to the non-extra-credit) to each problem as a **separate PDF** file (one PDF per problem) to codePost. Please make sure you are uploading the correct PDF! Please anonymize your submission (i.e., do not list your name in the PDF), but if you forget, it's OK.
- If you choose to do extra credit, upload your solution to the extra credits as a single separate PDF file to codePost. Please again anonymize your submission.
- You may collaborate with any classmates, textbooks, the Internet, etc. Please upload a brief "collaboration statement" listing any collaborators as a separate PDF on codePost (if you forget, it's OK). But always **write up your solutions individually**.
- For each problem, you should aim to keep your writeup below one page. For some problems, this may be infeasible, and for some problems you may write significantly less than a page. This is not a hard constraint, but part of the assignment is figuring out how to easily convince the grader of correctness, and to do so concisely. "One page" is just a guideline: if your solution is longer because you chose to use figures (or large margins, display math, etc.) that's fine.
- Each problem is worth twenty points (even those with multiple subparts), unless explicitly stated otherwise.

Problems:

§1 A cut is said to be a *B-approximate min cut* if the number of edges in it is at most B times that of the minimum cut. Show that all undirected graphs have at most $(2n)^{2B}$ cuts that are B-approximate.

Hint: Run Karger's algorithm until it has $2B$ supernodes. What is the chance that a particular B-approximate cut is still available? How many possible cuts does this collapsed graph have?

§2 Consider an unweighted, undirected simple graph $G = (V, E)$ whose mincut has value $c = \omega(\log n)$. You would like to create a *sparser* (weighted) graph $G' = (V, E')$ with $E' \subset E$, but such that the weight of each cut is approximately preserved. Specifically: sample every edge $e \in E$ with probability p. If e is sampled, add e to E' with weight $1/p$. You want that for all $S \subseteq V$, $\text{CUT}_{G'}(S) \in (1 \pm \varepsilon) \cdot \text{CUT}_G(S)$. The smaller you make p, the sparser G' will be (and therefore easier to store/operate/etc.). The bigger you make p the more likely you are to actually preserve the value of all cuts (consider e.g. $p = 1$).

Prove that for all constant ε, that for any graph G on n nodes with mincut $c = \omega(\log n)$, taking $p = O(\frac{\ln n}{\varepsilon^2 c})$ results with high probability (i.e., at least $1 - o(1)$) in a G' satisfying: for all $S \subseteq V$, $\mathrm{CUT}_{G'}(S) \in (1 \pm \varepsilon) \cdot \mathrm{CUT}_G(S)$.

Hint: The previous question might help!

§3 In class we saw that when hashing m items into a hash table of size $O(m^2)$, the expected number of collisions was < 1. In particular, this meant we could easily find a "perfect" hash function of the table that has no collisions.

Consider the following alternative scheme: build two tables, each of size $O(m^{1.5})$ and choose a separate hash function for each table independently. To insert an item, hash it to one bucket in each table and insert it only in the emptier bucket (tie-break lexicographically).

(a) Show that, if we're hashing m items, with probability 1/2, there will be no collisions in either table (a collision occurs when multiple distinct elements are inserted into the same bucket in the same table). You may assume access to a fully random hash function (see definition in lecture notes).

(b) Modify the above scheme to use $O(\log m)$ tables, each of size $O(m)$. Prove again that with probability 1/2, there will be no collisions in any table. Again, you may assume a fully random hash function.

§4 Let $X = \{x_1, \ldots, x_n\}$ be n (not necessarily distinct) unknown numbers in $[0, 1]$. You may repeatedly *sample with replacement* a uniformly random element of X.

(a) Prove that no algorithm using $o(n)$ samples can estimate the value of the median within a factor of 1.1. That is, any algorithm which (possibly randomly) maps $m = o(n)$ samples to a guess at the median is off by a factor of at least 1.1 with probability at least 1/3.

Hint: come up with two instances with very different medians, but which look the same after $o(n)$ samples with high probability. To prove that your algorithm must behave similarly on these instances, recall that your algorithm's behavior is completely determined by the samples it sees (you can avoid tedious calculations by cleverly using this observation).

(b) Instead, say we seek a number y such that at least $n/2 - t$ elements of X exceed y, and $n/2 - t$ numbers are less than y. Prove that if we take $m = O(n^2 \ln(1/\delta)/t^2)$ samples, and let y denote the median of the m samples, then y has this property with probability at least $1 - \delta$.

§5 (a) Consider the following random process: there are n coupons $\{1, \ldots, n\}$. Each step, you draw a uniformly random coupon independently with replacement, and you repeat this until you have drawn *all coupons in* $\{1, \ldots, n\}$. Prove that with probability at least $1 - 1/n$, the process takes $O(n \log n)$ steps.

Hint: You may find it easier to use Chernoff bounds and union bounds.

(b) Suppose that there are n items whose sizes $X_1, X_2, \ldots, X_n$ are drawn independently from $Unif[0, 1]$. In the (NP-Hard) bin packing problem, the goal is to

find the *minimum* number $f(X_1, X_2, \ldots, X_n) \in \mathbb{N}$ of bins such that it's possible to pack these items into f unit-sized bins (i.e., each item is assigned to a bin and the sum of sizes of items assigned to each bin is at most 1).
(i) Show that $\mu := \mathbb{E}[f]$ is $\Omega(n)$.
(ii) Show that for any $i \in \{1, \ldots, n\}$, if the size of i-th item changes then the value of f changes by at most 1. That is, prove that

$$|f(X_1, \ldots, X_i \ldots, X_n) - f(X_1, \ldots, X_i', \ldots X_n)| \leq 1,$$

where each $X_1, X_2, \ldots X_i, X_i', X_{i+1} \ldots, X_n$ is chosen *arbitrarily* in $[0, 1]$.

Remark: The above condition (ii) on f is known as Bounded-Difference condition. McDiarmid's concentration inequality is that if f satisfies (ii) then $\Pr_{X_1,\ldots,X_n}[f > (1+\epsilon)\mu] \leq \exp\left(\frac{\epsilon^2\mu}{10n}\right)$. Thus, even though computing f is NP-Hard for adversarial inputs, for random inputs it's w.h.p. only $\pm O(\sqrt{n})$ around its mean.

§6 Consider the following process for matching n jobs to n processors. In each step, every job picks a processor at random. The jobs that have no contention on the processors they picked get executed, and all the other jobs *back off* and then try again. Jobs only take one round of time to execute, so in every round all the processors are available. Show that all the jobs finish executing w.h.p. after $O(\log \log n)$ steps.

Hint: Try to argue that if there are currently ϵn unmatched jobs, then in the next round roughly $\epsilon^2 n$ jobs remain unmatched.

Extra Credit:

§1 (extra credit) Consider the following problem: there are $n > k$ independent (but not identically distributed) non-negative random variables $X_1, \ldots, X_n$ drawn according to distributions $D_1, \ldots, D_n$. Initially, you know each D_i but none of the X_is.

Starting from $i = 1$, each X_i is revealed one at a time. Immediately after it is revealed, you must decide whether to "accept i" or "reject i," before seeing the next X_{i+1}. You may accept at most k elements in total (that is, once you've accepted k times, you must reject everything that comes after). Your reward at the end is $\sum_{i|i \text{ was accepted}} X_i$.

(a) For general k, design a policy that guarantees expected reward at least $(1 - O(\sqrt{\frac{\ln(k)}{k}})) \cdot \mathbb{E}_{X_1,\ldots,X_n \leftarrow D_1,\ldots,D_n}[\sum_{j=1}^{k} X_{r(j)}]$, where r is a permutation from $[n]$ to $[n]$ satisfying $X_{r(1)} \geq X_{r(2)} \geq \ldots \geq X_{r(n)}$ (i.e. the policy gets expected reward at least $(1 - O(\sqrt{\frac{\ln(k)}{k}}))$ times the expected sum of top k weights, which is the best you could do even if you knew all the weights up front).
Hint: Try to set up a simple policy that can be analyzed using a Chernoff bound.

(b) Come up with an example showing that it is not possible to improve the above guarantee beyond $(1 - \Omega(1/\sqrt{k}))$ (which is optimal - no need to prove this).
Hint: an example exists whose complete proof should fit in half a page. You may use without proof the fact that if X is the number of coin flips which land heads from k independent fair coin flips, then $\mathbb{E}[|X - k/2|] = \Theta(\sqrt{k})$.