CSE 521: Design and Analysis of Algorithms **Fall 2021**

Problem Set 2

Deadline: Nov 2nd (at 11:59 PM) in gradescope

In solving these assignments and any future assignment, feel free to use these approximations:

$$1 - x \approx e^{-x}, \qquad n! \approx (n/e)^n, \qquad \left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k$$

1) In this problem we see how to use pairwise independent hash functions for de-randomization. Say A is a randomized algorithm that uses m random bits and will output the optimum solution of a minimization problem with probability $1/2$. In the first lecture we argued that we can improve the success probability to $1-1/2^k$ by simply running k *independent* copies of A and return the minimum outputted solution. But that needs $O(km)$ random bits. In practice random bits are a valuable resource and it is importance if you can minimize the use of this resource. In this question we want to see if it is still possible to increase the success probability of A to a value close to 1, without using too many random bits.

   Prove that for any $r \leq 2^m$, we can improve the success probability to $1 - 1/r$ using $O(m)$ random bits by running A only $O(r)$ many times. In other words, instead of using rm random bits that you would normally use in the above strategy, you want to use $O(m)$ bits but still improve the success probability to $1 - 1/r$ (as opposed to $1 - 2^{-r}$).

2) Consider the following streaming problem called (k, ℓ_1)-point query: Start with an n-dimensional zero vector $x = \mathbf{0}$. At any time t we are getting an update (i, Δ) where $1 \leq i \leq n$ and Δ is a (not-necessarily positive) integer and we update $x_i \leftarrow x_i + \Delta$. Note that you may update each coordinate multiple times (sometimes positively and sometimes negatively). At query time we are given an integer i and we must output x_i within a $\pm\|x\|_1/k$ additive error.

   Consider the following algorithm: Choose L many pairwise independent hash functions $h_1, \dots, h_L : [n] \to [4k]$. We store counters $C_{a,b}$ for all $a \in [L], b \in [4k]$. Upon an update (i, Δ), we add Δ to all counters $C_{a,h_a(i)}$, for $a \in [L]$. For a query i we simply output

$$\text{median}_{1 \leq a \leq L} C_{a,h_a(i)}$$

   a) Prove that, for a given $\delta > 0$ we can choose $L = O(\log(1/\delta))$ such that upon receiving any query i, the output C of the algorithm satisfies

$$\mathbb{P}\left[x_i - \frac{\|x\|_1}{k} \leq C \leq x_i + \frac{\|x\|_1}{k}\right] \geq 1 - \delta.$$

   b) How much memory does this algorithm need? You can assume that you need a single memory word to store each counter $C_{a,b}$.

3) a) Let $X_1, \dots, X_n$ be independent random variables uniformly distributed in $[0, 1]$ and let $Y = \min\{X_1, \dots, X_n\}$. Show that $\mathbb{E}[Y] = \frac{1}{n+1}$ and $\text{Var}(Y) \leq \frac{1}{(n+1)^2}$.

   Consider the following algorithm for estimating F_0, the number of unique elements in a sequence $x_1, \dots, x_m$ in the set $\{0, 1, \dots, n-1\}$. Let $h : \{0, 1, \dots, n-1\} \to [0, 1]$ s.t., $h(i)$ is chosen uniformly and independently at random in $[0, 1]$ for each i. We start with $Y = 1$. After reading each element x_i in the sequence we let $Y = \min\{Y, h(x_i)\}$.

   b) Show that by the end of the stream $\frac{1}{\mathbb{E}[Y]} - 1$ is equal to F_0.

c) Use the above idea to design a streaming algorithm to estimate the number of distinct elements in the sequence with multiplicative error $1 \pm \epsilon$. For the analysis you can assume that you have access to k independent hash functions as described above. Show that $k \leq O(1/\epsilon^2)$ many such hash functions is enough to estimate the number of distinct elements within $1 + \epsilon$ factor with probability at least 9/10.

4) In this problem we design an LSH for points in $\mathbb{R}^d$, with the ℓ_1 distance, i.e.

$$d(p, q) = \sum_i |p_i - q_i|.$$

a) Let a, b be arbitrary real numbers. Fix $w > 0$ and let $s \in [0, w)$ chosen uniformly at random. Show that

$$\mathbb{P}\left[\left\lfloor \frac{a-s}{w} \right\rfloor = \left\lfloor \frac{b-s}{w} \right\rfloor\right] = \max\left\{0, 1 - \frac{|a-b|}{w}\right\}.$$

Recall that for any real number c, $\lfloor c \rfloor$ is the largest integer which is at most c.

Hint: Start with the case where $a = 0$.

b) Define a class of hash functions as follows: Fix w larger than diameter of the space. Each hash function is defined via a choice of d independently selected random real numbers $s_1, s_2, \ldots, s_d$, each uniform in $[0, w)$. The hash function associated with this random set of choices is

$$h(x_1, \ldots, x_d) = \left(\left\lfloor \frac{x_1 - s_1}{w} \right\rfloor, \left\lfloor \frac{x_2 - s_2}{w} \right\rfloor, \ldots, \left\lfloor \frac{x_d - s_d}{w} \right\rfloor\right).$$

Let $\alpha_i = |p_i - q_i|$. What is the probability that $h(p) = h(q)$ in terms of the α_i values? For what values of p_1 and p_2 is this family of functions $(r, c \cdot r, p_1, p_2)$-sensitive? Do your calculations assuming that $1 - x$ is well approximated by e^{-x}.

5) We are given a directed graph $G = (V, E)$ with n vertices and m edges. For each vertex $v \in V$, let N_v denote the set of all nodes reachable from v (including v itself) and let $n_v = |N_v|$. Our goal is to find n_v for each vertex $v \in V$. The best exact algorithm for this problem runs in time $O(\min\{m \cdot n, n^{2.373}\})$. Here, we will explore a randomized approximation algorithms that runs in time $O(m \log n)$. The algorithm works as follows:

Step 1) For each vertex $v \in V$ choose a uniformly and independently random number $s_v \in [0, 1]$. Call this number the *minhash* of v.

Step 2) For each vertex $v \in V$, let $h_v := \min_{w \in N_v} s_w$. The h_v values for all vertices $v \in V$ can be found in time $O(m \log n)$!

Step 3) Let $\tilde{n}_v = \frac{1}{h_v}$.

Repeat all of the above steps k times and for each v output as your estimate for n_v, the median of $\tilde{n}_v$ values from the k runs.

a) Describe an implementation of step 2 that runs in $O(m \log n)$ time.

b) Implement the above algorithm for $k = 20, 200$ to approximate n_v for each vertex v in the vertex set of a directed graph. Implement also an exact algorithm for computing these n numbers. Run your algorithm on an input file that I will upload to the course website. The first line of the input indicates the number of nodes and directed edges and each following line indicates the source and destination of a directed edge in order.

6) **Extra Credit:** A random walker is walking on a circle with n vertices $\{1, \ldots, n\}$; it starts at vertex 1 and at each time step it moves to one of the two neighbors uniformly at random. It stops the moment that it visits every vertex at least once. What is the probability that the last vertex that it visits is $n/2$?