Homework 0

Due: N/A

This problem set is optional and will not be graded. Feel free to ask questions about these problems at hours! Solutions will be posted on January 31.

Please follow these guidelines when writing up your CS 155 homeworks.

A major goal of this class is that you learn how to mathematically analyze probabilistic processes and events. Hence, make sure your proofs are rigorous, i.e., that every step is adequately justified. Try to keep your proofs as simple and as concise as possible, keeping in mind that you will potentially have to examine multiple proof strategies in order to achieve this. Make sure to CLEARLY STATE YOUR ASSUMPTIONS at the beginning of any solution, when necessary. As we are doing probabilistic analysis, we shall require that you clearly define the probability space you are working with and identify the events within it that you will analyze.

You are allowed to discuss these problems with other students, but you have to leave any meeting with other students without any written material about the problems, and the writeups must be done by yourself without help from others. If you have any questions regarding how these guidelines apply to a particular problem or what they mean in general, please email the TAs.

Problem 1

The following approach is often called reservoir sampling. Suppose we have a sequence of items passing by one at a time. We want to maintain a sample of one item with the property that at each step it is uniformly distributed over all the items that we have seen up to this step. Moreover, we want to accomplish this without knowing the total number of items in advance or storing all of the items that we see.

Consider the following algorithm, which stores just one item in memory at all times. When the first item appears, it is stored in the memory. When the $k$th item appears, it replaces the item in memory with probability $1/k$. Explain why this algorithm solves the problem.
Problem 2

We have a function $F : \{0, \ldots, n - 1\} \to \{0, \ldots, m - 1\}$. We know that, for $0 \leq x, y \leq n - 1$,

$F((x + y) \mod n) = (F(x) + F(y)) \mod m.$

The only way we have for evaluating $F$ is to use a lookup table that stores the values of $F$. Unfortunately, an Evil Adversary has changed the value of $1/5$ of the table entries when we were not looking.

a. Describe a simple randomized algorithm that, given an input $z$, outputs a value that equals $F(z)$ with probability at least $1/2$. Your algorithm should work for every value of $z$, regardless of what values the Adversary changed. Your algorithm should use as few lookups and as little computation as possible.

b. Suppose I allow you to repeat your initial algorithm three times. What should you do in this case, and what is the probability that your enhanced algorithm returns the correct answer?

Fun fact: one application that this problem relates to is error-correcting codes. Suppose one party wants to send a string $s$ to another party, but the communication channel between them is noisy: if a message is sent over the channel, some of the bits might be flipped. So if the sender sends the string 110, the receiver might receive 100 instead.

One solution is to use an error-correcting code. Loosely speaking, an error-correcting code $C$ allows the sender to encode their message $s$ as a longer string $C(s)$ that is more robust against errors: even if some of the bits of $C(s)$ are flipped, it is possible to determine the original message $s$ from the corrupted version of $C(s)$. Thus, the sender can send $C(s)$ instead of $s$, and the receiver can use the (possibly corrupted) value received to recover $s$.

Now, suppose the string $C(s)$ represents the lookup table of a function $F$. (For example, if $m = 2$ in this problem, then the lookup table of $F$ can be represented by a string of length $n$, where the $(i + 1)$st bit is the value of $F(i)$.) This problem shows that if $F$ is a linear function (i.e., $F(x + y) = F(x) + F(y)$), then it is possible to recover $C(s)$ even if $1/5$ of its bits are corrupted.
Hence, if the sender and receiver agree on a way to associate a string $s$ with a linear function $F_s$, then encoding $s$ as the lookup table of $F_s$ results in an error-correcting code.

Problem 3

If you reload the home page of the course website a few times, you might notice that the picture of the mathematician changes. Ever wonder how many different pictures there are, and how you can find out?

Consider the following scenario. There are $n$ different pictures, where $n$ is unknown to you. Each time you load the website, it chooses one of the $n$ pictures uniformly at random and displays it.

Suppose we want to design an algorithm that outputs $n$. The algorithm can load the website and see the displayed picture (it cannot view the page source, etc.). For example, the following is a valid algorithm that loads the website 100 times and outputs the number of different pictures displayed:

\begin{algorithm}
\begin{enumerate}
\item Initialize $S$ to be the empty set. ($S$ will store the different pictures seen so far.)
\item Repeat 100 times:
\begin{enumerate}
\item Load the website.
\item If the displayed picture is not in $S$, then add it to $S$.
\end{enumerate}
\item Output the size of $S$.
\end{enumerate}
\end{algorithm}

This algorithm would probably find the right answer for our course website because we don’t have too many different pictures. But if $n$ were larger, say 50, then it’s not so clear how well this algorithm would work. And if $n$ were more than 100, then this algorithm cannot possibly succeed.

We want to design an algorithm that succeeds (outputs the correct value of $n$) with probability at least $1/2$, regardless of the value of $n$. Feel free to stop and think about how you might design such an algorithm. In the rest of this problem, we’ll describe one solution and ask you to analyze it.
The following algorithm keeps loading the website, keeping track of the different pictures that have been seen so far. If \( s \) is the number of different pictures that have been seen so far, the algorithm stops if no new pictures have been seen in the last \( 4s \) website visits.

### Algorithm.

1. Initialize \( S \) to be the empty set. (\( S \) will store the different pictures seen so far.)
2. \( s \leftarrow 0 \). (\( s \) represents the size of \( S \).)
3. \( Count \leftarrow -1 \).
4. Repeat until \( Count \geq 4s \):
   i. Load the website.
   ii. If the displayed picture is not in \( S \): add it to \( S \), \( s \leftarrow s + 1 \), and \( Count \leftarrow 0 \). Else, \( Count \leftarrow Count + 1 \).
5. Output the size of \( S \).

a. Assume \( s < n \). Given that at some point the algorithm has observed \( s \) different pictures, prove that the probability that it terminates before observing \( s + 1 \) pictures is bounded by

\[
(1 - \frac{n - s}{n})^{4s} \leq e^{-4s(n-s)/n}.
\]

Here is an inequality that might help in your analysis and will be useful throughout the course: for any real number \( x \),

\[ 1 - x \leq e^{-x}. \]

b. Prove that the probability that the algorithm does not give the correct answer is bounded by

\[
\sum_{s=1}^{n-1} e^{-4s(n-s)/n} \leq \sum_{s=1}^{n/2} e^{-2s} + \sum_{s=n/2+1}^{n-1} e^{-2(n-s)} \leq 1/2.
\]