CS91T Week 3 In-Lab Exercises

February 2, 2023

We will use labs in CS91T in several different ways:

- To introduce a bi-weekly lab assignment and get you started working on the assigment,
- To give one of the three tests this semester,
- To work on in-lab exercises.

This week we'll do in-lab exercise. When working on in-lab exercises, you'll be working on a number of problems in groups of 3-4 students each. You will not be handing in solutions. The primary purpose of in-lab exercises are to have a low-pressure space to discuss randomized algorithm design, and to gain experience collaborating with others on algorithm design and analysis.
The learning goals of lab exercises this week is to build up comfort and intuition of how an algorithm can use randomness, and to work with a CS application of randomness.

1. **Randomized Algorithms from Basic Primitives.** In this problem, assume a randomized algorithm is an algorithm that can use all the standard operations we've seen in CS21 and CS35 (e.g., if/else statements, for loops, math operations like +,-,*,/, logic operations like AND, OR, NOT, bitshift operations like $<<, >>, \ldots$), but additionally a single randomized operation flipCoin(), which takes no input and returns 1 with probability 0.5 and 0 with probability 0.5.

   (a) Design an algorithm Alg1 that takes no input and returns a uniform integer between $0 \ldots 7$ inclusive.

   (b) Design an algorithm Alg2 that takes a positive integer k as input and returns a uniform integer betweek $0 \ldots (2^k - 1)$ inclusive.

   (c) Design an algorithm Alg3 that takes a positive integer n as input and returns a uniform integer between $1 \ldots n$.

   (d) Design an algorithm Alg4 that takes a positive integer n as input and returns a uniform *subset* $S \subseteq \{1, \ldots, n\}$

2. **Emprical Analysis of Random Number Generators**. In this lab exercise, you'll use the random number generators given in Python and C++ and write programs that examine how random the random number generators actually are in practice.

   For the most part this semester, we'll assume that the randomness we use is completely uniform (or has some other well-defined distribution), but actually simulating this in practice is a challenging problem beyond the scope of CS91T. Understanding how real-world random number generators work might be a good final project topic.

(a) Write a program that, for a given n, samples n random bits and outputs the number of ones that were sampled. Then, run your program several times. Does the random number generator you're using seem to be uniform? How does the number of heads scale with n? If the number generator is truly uniform, you should expect to generate $n/2 \pm O(\sqrt{n})$ heads.[1]

(b) Write a second program that, for a given n and k, samples n random elements from the range $1 \dots k$. For each $1 \leq i \leq k$, count and output the number of times you generated an i. Does the random number generator you're using seem to be uniform? How does the fraction of samples that are i change as you scale with n?

3. **Communication Complexity of Equality**. Alice and Bob each have inputs and want to talk to decide if their inputs are equal. Unfortunately, they're friendship has hit a rocky spot, and they'd like to do this while communicating as little as possible. How much do they need to communicate to determine if their inputs are equal? Call this the EQUALITY problem.

   To formalize this, suppose that Alice's input is a bitstring $x \in \{0,1\}^n$. Bob's input is similarly $y \in \{0,1\}^n$, and their goal is to compute $\text{EQ}(x,y)$, which equals 1 if $x == y$ and equals 0 if $x \neq y$.

   Say the deterministic communication complexity of EQUALITY is the minimum number of bits of communication required to compute $\text{EQ}(x,y)$ of any strategy that Alice and Bob use that doesn't use randomness. Similarly, the randomized communication complexity of EQUALITY is the minimum number of bits of communication of a randomized strategy of computing $\text{EQ}(x,y)$.

   (a) Suppose that the communication consists of a single message from Alice to Bob, and that the message is generated without using randomness. Show that Alice's message must be at least n-bits long.

   (b) Now, suppose that Alice and Bob have joint access to a source of randomness FLIPCOIN(). (note: they both see the random bits that are sampled from FLIPCOIN()) Furthermore, suppose that Alice and Bob are ok if they sometimes get the wrong answer: for any pair of possible inputs (x,y), their strategy should correctly compute $\text{EQ}(x,y)$ with probability at least 0.99.

   Design and analyze a strategy for computing $\text{EQ}(x,y)$ that uses $O(1)$ bits of communication.

   (c) Show that the deterministic communication complexity of EQ is n bits, even if Alice and Bob are allowed to communicate back and forth.

[1] You will see how to rigorously analyze this soon, but we need a *bit* more probability first.