COMPSCI 690RA: Problem Set 1

Due: 2/15 by 8pm in Gradescope.

Instructions:

- You are allowed to, and highly encouraged to, work on this problem set in a group of up to three members.
- Each group should **submit a single solution set**: one member should upload a pdf to Gradescope, marking the other members as part of their group in Gradescope.
- You may talk to members of other groups at a high level about the problems but **not work through the solutions in detail together**.
- You must show your work/derive any answers as part of the solutions to receive full credit.

Hint: The following two inequalities may be helpful in various places (and generally throughout the course): for any $x > 0$, $(1+x)^{1/x} \leq e$ and $(1-x)^{1/x} \leq 1/e$.

1. Probability Practice (6 points)

1. (2 points) Consider storing n items in a hash table with $m = n$ buckets, using a fully random hash function $\mathbf{h} : [n] \to [n]$ (i.e., each item is assigned independently to a uniform random bucket). What is the expected load factor of the hash table? I.e., what is the expected fraction of buckets that have at least one item in them? What is the limit of this value as $n \to \infty$? What about when $m = 2n$? **Hint:** Use linearity of expectation.

2. (2 points) Consider a random walk on a d-dimensional grid. At each step, the walk chooses one of the d-dimensions uniformly at random, and takes a step in that direction – up with probability $1/2$ and down with probability $1/2$. Assume that the walk starts at the origin and ends at position $x \in \mathbb{Z}^d$ and takes n steps. What is $\mathbb{E}[\|x\|_2^2]$, where $\|x\|_2^2$ denotes the squared Euclidean norm of x.

3. (2 points) A monkey types on a 26-letter keyboard that has lowercase letters only. Each letter is chosen independently and uniformly at random from the alphabet. If the monkey types 100,000,000 letters. what is the expected number of times the sequence 'random' appears? Give an upper bound on the probability that 'random' appears at least once.

2. Translating Between Randomized Guarantees (4 points)

1. (2 points) Suppose I have a Las Vegas algorithm that solves a decision problem with expected runtime T. For any $\delta > 0$, describe and analyze a Monte-Carlo algorithm that correctly answers the decision problem with probability at least $1 - \delta$ and has worst case runtime $O(\log(1/\delta) \cdot T)$.

2. (2 points) Suppose I have a Monte-Carlo algorithm that solves a decision problem with worst case runtime T and at least $2/3$ probability of correctness. For any $\delta > 0$, describe and analyze a Monte-Carlo algorithm that correctly answers the decision problem with probability at least $1 - \delta$ and has worst case runtime $O(\log(1/\delta) \cdot T)$.

3. Finding Random Primes (4 points)

A common application of randomized primality testing algorithms is in finding large primes for use in cryptographic schemes, such as RSA. Suppose you have a Monte-Carlo primality testing algorithm that, given an integer input x, outputs 'yes' or 'no'. If x is prime, the algorithm always outputs 'yes'. If x is composite, the algorithm outputs 'no' with probability at least $1/2$.

For any $\delta \geq 0$, describe an algorithm that makes $O(\log(n) \cdot \log(1/\delta))$ calls to this tester and outputs x such that, with probability at least $1 - \delta$, x is a uniformly random prime in $\{1, \ldots, n\}$.

Hint 1: Let $\pi(n)$ be the *prime counting function*: the number of prime numbers less than or equal to any integer n. You may use the fact that for any $n \geq 17$, $\frac{n}{\log n} \leq \pi(n)$. This fact is closely related to the prime number theorem (PNT).

Hint 2: It might be easier to first shoot for $O(\log(n) \cdot \log(1/\delta)^2)$ calls and then figure out how to tighten your analysis.

4. Randomized Routing (4 points)

Consider the following simplified model of a routing problem under communication constraints: we have n nodes in a fully connected network. Each node has n messages that it would like to deliver to n (not necessarily unique) destinations. Also, for simplicity, we assume that each node is the intended recipient of exactly n messages. In each round, a node can send at most one message along each connection in the network. Assume that the message contains the id of its final intended recipient. Naively, sending these messages could take n rounds, if e.g., some node v needs to send n messages to some other node u.

Describe and analyze a randomized scheme that sends all messages in $O(\log n)$ rounds with high probability, i.e., with probability at least $1 - 1/n^c$ for a large constant c.

Hint: Have each node initially send each of its n messages to a random recipient, who will then forward the message to its final destination.

To think about: Can you do better than $O(\log n)$ rounds? What can you achieve deterministically?

5. Building on Freivald's Algorithm (4 points)

Consider two matrices $A, B \in \mathbb{R}^{n \times n}$ whose product AB has just k non-zero entries. Describe a randomized algorithm that computes AB in $O(n^2 \cdot k \cdot \log n)$ time, with probability at least $99/100$.

Hint: Use repeated applications of Freivald's approach to determine which columns of AB are non-zero. Some students pointed out a much simpler and faster alternative to my solution that runs in $O(n^2(\log n + k))$ time. I think it is more natural, so don't be surprised if you get that runtime.

Bonus: (2 points) Give an algorithm that runs in $O(nk^2 + n^2 \log n)$ time, which for the interesting case of $k < n$ is faster than both my runtime and the students' runtime stated above. 3 points awarded if you achieve runtime $O(nk + n^2 \log n)$, which is even better. I believe this may be possible but am not sure.

6. Stacking Hash Tables (8 points)

1. (3 points) Consider storing n items in a hash table with m buckets, using a fully random hash function $\mathbf{h} : [n] \to [m]$ (i.e., each item is assigned independently to a uniform random bucket). Prove that if $m = cn^2$ for some sufficiently large constant c, then there are *no collisions* in the hash table with probability at least $9/10$. Thus, the table has worst case $O(1)$ lookup time. **Hint:** Use linearity of expectation, considering all pairs of items (i, j) that may collide.

2. (3 points) Consider storing n items in two hash tables with m buckets each, using two different fully random hash functions $\mathbf{h}_1 : [n] \to [m]$, $\mathbf{h}_2 : [n] \to [m]$. An item x is stored in bucket $\mathbf{h}_1(x)$ of the first table, unless this bucket already has an item in it. In that case, it is stored in bucket $\mathbf{h}_2(x)$ of the second table. How large must m be such that, with probability at least $9/10$, there are no collisions in this scheme? I.e., so that all buckets in both the first and second tables have at most 1 item in them. What is the worst case lookup time for this scheme?

3. (2 points) Generalize the above approach to use any constant number of hash tables $t \geq 1$. What is the tradeoff between the amount of storage you must allocate to the hash tables and the worst-case lookup time?