Ground Rules

See homework 1.

Ungraded problems

1. The bias of a coin is the probability of obtaining heads in a single toss of the coin. In algorithm design, we are often interested in coins of bias $1/2$, but natural sources of randomness (such as radioactive decay) can be quite skewed.

   (a) Given a coin $C$ of bias $p$, give an algorithm for generating a coin toss with bias $1/2$, using as few tosses of $C$ in expectation as possible. Can you design an algorithm that does not require knowledge of $p$?

   (b) Sometimes we have available a coin of bias $1/2$, but require a coin of bias $p$ for some value $p < 1/2$. Give an algorithm for generating a coin toss of bias $p$ using a coin $C$ with bias $1/2$ using as few tosses of $C$ in expectation as possible.

2. Given a fair coin, that is a coin of bias $1/2$, how would you generate a random number between 1 and $n$, all numbers being equally likely? How many coin tosses do you need?

3. Problem 13.9 in the textbook (Pg. 788-789). Note that it is possible to get a success probability better than $1/4$ but you only need to prove a $1/4$ for this homework.

Graded problems

4. (5 points) Problem 13.8 in the textbook (Pg. 788).

   Update: You are only required to construct an algorithm that always runs in polynomial time and returns a solution with the expected number of edges being at least $\frac{mk(k-1)}{n(n-1)}$. Producing a solution that always obtains at least $\frac{mk(k-1)}{n(n-1)}$ edges will earn an extra credit.

5. (5 points) A Bloom filter is like a hash table, but without linked lists to take care of collisions. Here is how it works. We allocate a binary array $A$ of size $m$; Each position in the array is initially 0. We also pick $k$ hash functions $h_1, \cdots, h_k$, each mapping the universe $U$ to indices in $\{1, \cdots, m\}$. To insert an element $i \in U$ in the Bloom filter, we compute the $k$ indices, $h_1(i), h_2(i), \cdots, h_k(i)$, and write a “1” in the array $A$ at each of those indices. To determine whether an element $i$ is in the Bloom filter or not, we compute the $k$ indices, $h_1(i), h_2(i), \cdots, h_k(i)$, and return a “Yes” answer if each of the corresponding positions in $A$ is 1, and otherwise return a “No” answer.

   For the purpose of this question you may assume that each hash function $h_j$ maps each element independently to a uniformly random position in the array. We will also not worry about deletions.

   Suppose that we insert a set $S$ of $n$ elements into the Bloom filter. Determine an expression for the probability that a lookup for a new element $i \notin S$ returns an incorrect answer. That is, the lookup algorithm says “Yes” even though we did not previously insert the element into the array$^1$.

   Compute this probability for $m = 50$, $k = 2$, and $n = 10$. Fixing $n = 10$, find values for $m$ and $k$ for which this probability is at most 2%. Strive for small values.

$^1$This kind of error is called a false positive. Note that Bloom filters do not return false negatives.
6. (3+2 points.) Randomness can be very useful when we want to verify the correctness of a program. In this question we will see a basic example of how randomness helps in generating good test cases. Suppose we are given $n \times n$ matrices $A$, $B$, and $C$, and wish to check whether $AB = C$. One way to do this is to just multiply $A$ and $B$ out and compare the result to $C$. This would take $O(n^3)$ time. We will now develop an $O(n^2)$ time test that has high accuracy.

(a) Let $r$ be an $n \times 1$ random vector, where each entry is independently 0 with probability $1/2$ and 1 with probability $1/2$. Prove that for two $n \times n$ matrices $X$ and $Y$ with $X \neq Y$, the probability that $Xr = Yr$ is at most $1/2$. Note that $X$ and $Y$ are fixed matrices, not random matrices.

**Hint:** Consider the non-zero matrix $(X - Y)$ and prove that $\Pr[(X - Y)r = 0]$ is less than $1/2$.

(b) How much time does it take to determine the product $Cr$ where $C$ is an $n \times n$ matrix and $r$ is an $n \times 1$ vector? What about the product $ABr$?

Use your observation and part (a) to design an $O(n^2)$ time randomized algorithm for checking whether $AB = C$, that returns the correct answer with probability at least $1/2$. More precisely, if $AB = C$ is correct, your algorithm should output “correct” with probability 1, and if it is not, then your algorithm should output “incorrect” with probability at least $1/2$.

\[\text{Or } O(n^{2.81}) \text{ time using fast matrix multiplication (Strassen’s algorithm).}\]