CS 161
Summer 2017

Problem Set 3
Due: July 21, 2017, 11:59 p.m.

---

1. **Some random hotel.** (2 points) There's a hotel with 100 rooms, each belonging to one of 100 guests. After an evening soiree, all of the guests (rather inebriated) randomly selects a room to sleep in for that night. Multiple guests might end up in the same room.

   (a) (1 point) What is the expected number of guests that end up returning to their own hotel room?

   [We are expecting a number. Please show your work.]

   (b) (1 point) What is the expected number of guests that end up in a room with exactly one other person?

   [We are expecting a mathematical expression like (6)(8) **or a number like** 48**. Please show your work.]**

2. **An exhausting set of hash functions.** (6 points) In this problem, we'll investigate the definition of universal hash functions. Let $\mathcal{U}$ denote a universe of size M, and let n be the number of buckets in a hash table.

   (a) (3 points) Let $\mathcal{H}$ be the family of all possible functions mapping from $\mathcal{U}$ to $\{1, ..., n\}$. Prove that, for all $x_i \neq x_j$ in $\mathcal{U}$, for h randomly chosen from $\mathcal{H}$,

   $$\Pr[h(x_i) = h(x_j)] = \frac{1}{n}.$$

   This shows that $\mathcal{H}$ is a universal hash family, and moreover that we have *equality* in the definition of the universal hash family property, not just a $\leq$ relationship.

   [We are expecting: a careful and rigorous proof, though it should not need to be more than one or two paragraphs. You should be especially careful about what is random and what is not.]

   (b) (3 points) There also exist hash families such that, for all $x_i \neq x_j$ in $\mathcal{U}$, for h randomly drawn from the family, $\Pr[h(x_i) = h(x_j)] < \frac{1}{n}$. (Notice that the inequality here is strict!) We will now explore one such family. Consider $\mathcal{H}' = \mathcal{H} \setminus \{h_1\}$, where h_1 is the function defined by

   $$h_1(x_i) = 1 \qquad \text{for all} \qquad x_i \in \mathcal{U}.$$

   That is, $\mathcal{H}'$ is the family of all functions from $\mathcal{U}$ to $\{1, ..., n\}$ *except* for the function h_1 which sends all elements of $\mathcal{U}$ to 1. Prove that, for all $x_i \neq x_j$ in $\mathcal{U}$, for h drawn randomly from $\mathcal{H}'$, we have

   $$\Pr[h(x_i) = h(x_j)] < \frac{1}{n}.$$

   [We are expecting: a clear and rigorous proof. As above, this needn't be more than a paragraph of two, but you should be very careful about what is random and what is not.]

(c) (1 point extra credit) Give a hash family $\mathcal{H}$ so that

$$\max_{x \neq y \in \mathcal{U}} \Pr[h(x) = h(y)]$$

is as small as possible (and ideally prove that it is as small as possible). What is this probability? Above, the probability is over the choice of a uniformly random $h \in \mathcal{H}$.

[We aren't expecting anything, this is a bonus problem.]

3. **Quick password recovery.** (7 points) Suppose that on your computer you have stored n password-protected files, each with a unique password. You've written down all of these n passwords, but you do not know which password unlocks which file. You've put these files into an array F and their passwords into an array P in an arbitrary order (so $P[i]$ does not necessarily unlock $F[i]$). If you test password $P[i]$ on file $F[j]$, one of three things will happen:

   1) $P[i]$ unlocks $F[j]$
   2) The computer tells you that $P[i]$ is lexicographically smaller than $F[j]$'s true password
   3) The computer tells you that $P[i]$ is lexicographically greater than $F[j]$'s true password

   You **cannot** test whether a password is lexicographically smaller or greater than another password, and you **cannot** test whether a file's password is lexicographically smaller or greater than another file's password.

   (a) (3 points) Design an randomized algorithm to match each file to its password, which runs in expected runtime $O(n \log(n))$.

   [We are expecting pseudocode and/or an English description of an algorithm.]

   (b) (2 points) Explain why your algorithm is correct.

   [We are expecting an informal argument (a paragraph or so) about why your algorithm is correct, which is enough to convince the reader/grader. You may also submit a formal proof if you prefer.]

   (c) (2 points) Analyze the running time of your algorithm, and show that it runs in expected runtime $O(n \log(n))$.

   [We are expecting a formal analysis of the runtime.]

4. **Bloom filters.** (5 points) Hash functions are extremely good at what they do. Unsurprisingly, there are many fancier data structures that can be built on top of them. In this problem we will motivate and explore the idea of a "Bloom Filter", which is one example of a fancier structure built on top of hash functions. (Feel free to Google around for resources on Bloom Filters if you are inspired.)

   Suppose you are hired by someone to make a plagiarism detection software for internal use so as to avoid any potentially embarrassing allegations of plagiarism. Specifically, your goal is to make a lightweight (i.e. fast, and relatively low-memory) piece of software that

will take a sentence and output one of the following messages: 1) "potentially problematic, please rewrite", or 2) "fresh like an ocean breeze." Suppose your goal is the following: if the input sentence is something that you have already seen, you output "potentially problematic" (with probability 1), and if the input is something new, you want to output "fresh" with probability at least 0.99 (its alright if you have a few false-alarms).

(a) (1 point) First, you decide to use a hash table. You will make a hash table that maps a piece of text to a bucket, then scrape the web for all English sentences, and hash each one to your table. Given a new sentence, you will check to see if it hashes to an empty bucket—if so, you will output option "fresh" otherwise you will output "potential plagiarism." Suppose there are 1 Billion unique sentences online—how many buckets will your hash table need to have to have the desired functionality?

[We are expecting a number and one to two sentences of justification.]

(b) (2 points) You decide that is a little too much space usage, and consider the following approach: you choose 10 hash functions, $h_1, \ldots, h_{10}$ that each map sentences to the numbers 1 though 10 billion. You initialize an array A of 10 billion bits, initially set to 0. For each sentence s that you encounter, you compute $h_1(s), h_2(s), \ldots, h_{10}(s)$, and set the corresponding indices of A to be 1 (namely you set $A[h_1(s)] := 1, A[h_2(s)] := 1, \ldots$). Argue that after processing the 1 Billion unique sentences, you expect a $(1 - 1/(10 \text{ billion}))^{10\text{billion}} \approx 0.37$ fraction of the elements to be 0.

For this part, feel free to assume that the h_i are "idealized" hash functions that map each key s to a uniformly random bucket.

[We are expecting a paragraph with your argument.]

(c) (2 points) Now, given a sentence s, to check if it might be plagiarized, you compute the 10 hashes of s, and check if $A[h_1(s)] = A[h_2(s)] = \ldots = A[h_{10}(s)] = 1$. If so, you output "potential problem," otherwise you output "fresh." Prove that if s is actually in your set of 1 Billion sentences, that you will output "potential problem" with probability 1, and that if s is *not* in your set of 1 Billion sentences, you will output "fresh" with probability $\approx 1 - (1 - 0.37)^{10} \approx 0.99$. [Note: again, feel free to assume that the hash functions are random, and that the claim of the previous part holds, namely that after processing the 1 Billion sentences, there are 3.7 Billion indices in the array A with value 0.]

[We are expecting informal mathematical justifications for each of the bounds.]

(d) (0 points, food for thought) Is there a better tradeoff between the number of hash functions (10), and the size of A (10 Billion)? Specifically, is there a number of hash functions, t, such that with an array A of size less than 10 Billion you could still achieve a similar probability of correctly identifying "fresh" sentences?

5. **Colinear points.** (7 points)

You are given n distinct ordered pairs of integers $(x_1, y_1), (x_2, y_2), ..., (x_3, y_3)$, where for all i, j, $x_i \neq x_j$ and $y_i \neq y_j$. Recall two points uniquely define a line $y = mx + b$, with slope m and intercept b. We say a set of points S is **collinear** if they all fall on the same line; that is, for all $(x_i, y_i) \in S$, $y_i = mx_i + b$ for some fixed m and b. We want you to find the maximum cardinality subset of the given points A which are collinear. Assume that given two points, you can compute the corresponding m and b for the line passing through them in constant time, and you can compare two slopes or two intercepts in constant time. Your algorithms should not use any form of hash table.

(a) (2 points) Design an algorithm to find a maximum cardinality set of collinear points in $O(n^2 \log n)$ time. If there are several maximal sets, your algorithm can output any such set. Briefly justify the runtime of the algorithm.

[We are expecting a description (pseudocode optional) of your algorithm and a brief justification of its runtime.]

(b) (4 points) It is not known whether we can solve this collinear points problem in under $O(n^2)$ time. But suppose we know that our maximum cardinality set of collinear points consists of exactly n/k points for some constant k. Design a randomized algorithm that reports the points in some maximum cardinality set in expected time $O(n)$. (Hint: your running time may also be expressed as $O(k^2 n)$). Briefly justify the correctness and runtime of the algorithm.

[We are expecting a description (pseudocode optional) of your algorithm and a brief justification of its correctness and runtime.]

(c) (1 point) Is your algorithm from (b) guaranteed to terminate?

[We are expecting a Yes/No.]

6. **How is the course so far?** (0.5 points extra credit)

Please complete the poll at `https://goo.gl/forms/zD2ovjb3akeZdSKC3`.