New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6763: Homework 1.
Due Monday, September 27th, 2021, 11:59pm ET.

Collaboration is allowed on this problem set, but solutions must be written-up individually. Please list collaborators for each problem separately, or write "No Collaborators" if you worked alone.

For just this first problem set, 10% extra credit will be given if solutions are typewritten (using LaTeX, Markdown, or some other mathematical formatting program).

Problem 1: Collision free hashing.

(15 pts) Consider inserting m unique keys into a hash table of size $n = m^2$ using a uniformly random hash function. Using the mark-and-recapture analysis from class, we know that the expected number of collisions in the table is $\frac{m\cdot(m-1)}{2m^2} < 1/2$. So, by Markov's inequality, with probability $> 1/2$, the table has no collisions (strictly < 1 collisions). Thus, we can look up items from the table in worst-case $O(1)$ time.

1. Give an alternative proof of the fact that we have no collisions with $> 1/2$ probability in a table of size cm^2 for some sufficiently large constant c. Specifically, to have no collisions, we must have the following events all happen in sequence: the second item inserted into the hash table doesn't collide with an existing item, the third item inserted doesn't collide with an existing item, ..., the m^{th} item inserted doesn't collide with an existing item. Analyze the probability these events all happen. **Hint:** You might want to use the standard fact that $\frac{1}{2e} \leq (1 - \frac{1}{n})^n \leq \frac{1}{e}$ for any positive integer $n \geq 2$.

2. Consider the following alternative scheme: build two tables, each of size $O(m^{1.5})$ and choose a separate random hash function (independently at random) for each table. To insert an item, hash it to one bucket in each table and place it in the emptier bucket. Show that, if we're hashing m items, then with probability $\geq 1/2$, there will be no collisions in either table. You may assume uniformly random hash functions.

3. Modify the above scheme to use $O(\log m)$ tables. Prove that this approach yields a hashing scheme with space $O(m \log m)$ and $O(\log m)$ worst-case look-up time.

Problem 2: Why does Count-Min work well in practice?

(15 pts) We showed that Count-Min can estimate the frequency of any item in a stream of n items up to additive error $\frac{1}{m}n$ using $O(m)$ space. In practice it is often observed that this bound is pessimistic: the algorithm performs better than expected. In this problem, you will establish one reason why.

For any positive integer m, let $f_1, \ldots, f_m$ be the frequencies of the m most frequent items in our stream. Let $C = n - \sum_{i=1}^{m} f_i$. In general, we can have that $C \ll n$. For example, it has been reported that up to 95% of YouTube video views come from just 1% of videos. Prove that using $O(m)$ space, Count-Min can actually return an estimate $\tilde{f}$ to $f(v)$ for any item v satisfying:

$$f(v) \leq \tilde{f} \leq f(v) + \frac{1}{m}C$$

with 9/10 probability. This is strictly better than the $\frac{1}{m}n$ error bound shown in class.

Problem 3: Randomized methods for COVID-19 group testing.

(10 pts) One of the most important factors in controlling diseases like COVID-19 is testing. Unfortunately, testing can be expensive and slow. One way to make it cheaper is by testing patients in *groups*. The biological samples from multiple patients (e.g., multiple nose swabs) are combined into single test tube and tested for COVID-19 all at once. If the test comes back negative, we know everyone in the group is negative. If the test comes back positive, we do not know which patients in the group actually had COVID-19, so further testing would be necessary. There's a trade-off here, but it turns out that, overall, group testing can save on the total number of tests run.

1. Consider the following deterministic "two-level" testing scheme. We divide a population of n individuals to be tested into C arbitrary groups of the same size. We then test each of these groups in aggregate. For any group that comes back positive, we retest all members of the group individually. Show that there is a choice for C such that, if k individuals in the population have COVID-19, we can find all of those individuals with $\leq 2\sqrt{nk}$ tests. You can assume k is known in advance (often it can be estimated accurately from the positive rate of prior tests). This is already an improvement on the naive n tests when $k < 25\% \cdot n$.

2. We can use randomness to do better. Consider the following scheme: Collect $q = O(\log n)$ nose swabs from each individual (I know... not pleasant). Then, repeat the following process q times: randomly partition our set of n individuals into C groups, and test each group in aggregate. Once this process is complete, report that an individual "has COVID" if the group they were part of tested positive all q times. Report that an individual "is clear" if any of the groups they were part of tested negative. Show that for $C = O(k)$, with probability $9/10$, this scheme finds all truly positive patients and reports no false positives. Thus, we only require $O(k \log n)$ tests!

3. **Extra Credit – optional.** Show that no scheme can use $o(k \log(n/k))$ tests and succeed with probability $> 2/3$. So, for small k, the approach above is essentially optimal up to constant factors!

Problem 4: Hashing around the clock.

(15 pts) In modern systems, hashing is often used to distribute data items or computational tasks to a collection of servers. What happens when a server is added or removed from a system? Most hash functions, including those discussed in class, are tailored to the number of servers, n, and would change completely if n changes. This would require rehashing and moving all of our m data items, an expensive operation.

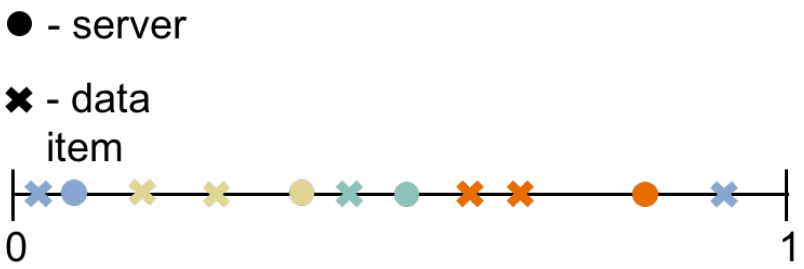

Figure 1: Each data item is stored on the server with matching color.

Here we consider an approach to avoid this problem. Assume we have access to a completely random hash function that maps any value x to a real value $h(x) \in [0, 1]$. Use the hash function to map *both* data items and servers randomly to $[0, 1]$. Each data item is stored on the first server to its right on the number line (with wrap around – i.e. a job hashed below 1 but above all serves is assigned to the first server after 0). When a new server is added to the system, we hash it to $[0, 1]$ and move data items accordingly.

1. Suppose we have n servers initially. When a new server is added to the system, what is the expected number of data items that need to be relocated?

2. Show that, with probability $> 9/10$, no server "owns" more than an $O(\log n/n)$ fraction of the interval $[0, 1]$. **Hint:** This can be proven without a concentration bound.

3. Show that if we have n servers and m items and $m > n$, the maximum load on any server is no more than $O(\frac{m}{n} \log n)$ with probability $> 9/10$.

Problem 5: Understanding Moment Bounds

(10 pts) Let X be a random variable uniformly distributed in the interval $[0, 1]$. Since we know X's distribution exacty, we can easily check that $\Pr[X \geq 7/8] = 1/8$. But let's take a look at what various concentration inequalities would predict about this probability using less information about X.

1. Given an upper bound on $\Pr[X \geq 7/8]$ using Markov's inequality.

2. Give an upper bound on $\Pr[X \geq 7/8]$ using Chebyshev's inequality.

3. Given an upper bound on $\Pr[X \geq 7/8]$ by applying Markov's inequality to the random variables X^2 (the "raw" second moment). Note that this is slightly different than using Chebyshev's inequality, which applies Markov to "central" second moment $(X - \mathbb{E}[X])^2$.

4. What happens for higher moments? Try applying Markov's to X^q for $q = 3, 4, \ldots, 10$. Describe what you see in a sentence. What value of q gives the tightest bound?

5. One take-away here is that, depending on the random variable being studied, it is not always optimal to use the variance as a deviation measure to show concentration. Markov's can be used with any monotonic function g, and as we see above, different choices might give better bounds. Exhibit a monotonic function g so that applying Markov's to $g(X)$ gives as tight an upper bound on $\Pr[X \geq 7/8]$ as you can. Maximum points if you can get $\Pr[X \geq 7/8] \leq 1/8$, which would be the best possible.