Recap of this week’s lectures:

- Hashing: Universal hashing, and constructions.
- Perfect hashing: dictionary lookup in constant worst-case time.
- The Data Streaming model
- Heavy hitters, both without and with deletions.

**Hashing:** A universal hash family $H$ from $U$ to $[m] := \{0, 1, \ldots, m - 1\}$ is a set of hash functions $H = \{h_1, h_2, \ldots, h_k\}$ each mapping $U$ to $[m]$, such that for any $a \neq b \in U$, when you pick a random function from $H$,

$$\Pr[h(a) = h(b)] \leq \frac{1}{m}.$$  

Also, a $\ell$-universal hash family $H$ from $U$ to $[m] := \{0, 1, \ldots, m - 1\}$ is a set of hash functions $H = \{h_1, h_2, \ldots, h_k\}$ each mapping $U$ to $[m]$, such that for any distinct $a_1, \ldots, a_\ell \in U$, and for any $\alpha_1, \ldots, \alpha_\ell \in [m]$, when you pick a random function from $H$,

$$\Pr[h(a_1) = \alpha_1 \text{ and } \ldots \text{ and } h(a_\ell) = \alpha_\ell] = \frac{1}{m^\ell}.$$  

1. Show that a 2-universal hash family is a universal hash family.

2. Show that a $k$-universal hash family is a $\ell$-universal hash family for any $\ell \leq k$.

3. Is this hash family from $U = \{a, b\}$ to $\{0, 1\}$ (i.e., $m = 2$) universal? 1-universal? 2-universal?

\[
\begin{array}{c|cc}
 & a & b \\
\hline
h_1 & 0 & 0 \\
h_2 & 1 & 0 \\
\end{array}
\]
Las Vegas and Monte Carlo:

A Las Vegas algorithm is a randomized algorithm that always produces the correct answer, but its running time $T(n)$ is a random variable. That is, sometimes it runs faster and sometimes slower, based on its random choices; for instance, quicksort when choosing a random pivot, or treaps. A Monte Carlo algorithm is an algorithm with a deterministic running time, but that sometimes doesn’t produce the correct answer. For example, you may be familiar with randomized primality testing algorithms, that given a number $N$ will output whether it is prime or not and be correct with probability at least $99/100$, say.

1. **Going from LV to MC:** Show that if you have a Las Vegas algorithm with expected running time $\mathbb{E}[T(n)] \leq f(n)$, then you can get a Monte Carlo algorithm with (a) worst-case running time at most $4f(n)$ and (b) probability of success at least $3/4$.

2. **Going from MC to LV:** Suppose you have a MC algorithm $\text{Algo}$ for a problem (e.g., think of factoring an $n$-bit number into a product of primes) with running time at most $f(n)$, that is correct with probability $p$. Moreover, you have a “checking” algo $\text{Check}$ that runs in time $g(n)$ and checks whether a given output is a correct solution for this problem. E.g., for factoring, you could just multiply the outputs together to make sure you get back the input, and also verify the outputs are indeed prime numbers by running a fast primality checker.

Use these to get a LV algorithm that runs in expected time $\frac{1}{p}(f(n) + g(n))$. 

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$h_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$h_3$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$h_4$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Streaming.

**Sampling:** Given a number $k$, you want to maintain a random sample of size $k$ from the stream. I.e., for each $n \geq k$, the set you have at time $n$ should be a random subset of the prefix $a[1:n]$, each of the $\binom{n}{k}$ subsets of size $k$ from this prefix should be equally likely.

1. For $k = 1$, show that the algorithm: pick the first element. When faced with the $n^{th}$ element, with prob. $1/n$ discard the element in your hand and pick the new element, and with prob. $1 - 1/n$ keep the element in hand.

2. Give an algorithm for general $k$. (What would you do when faced with the $n^{th}$ element? With what probability should you pick this element? Which element should you drop?)

**Missing Numbers:** Suppose I give you a stream of $n - 1$ elements, which contains all the numbers from 1 thru $n$ except one of them. (The numbers do not appear in sorted order.) Clearly you can figure out the missing number by storing all $n - 1$ numbers and looking for the missing number. How can you output the missing number with only $O(\log n)$ space? What if there are two missing numbers: can you again use only $O(\log n)$ space?