15-451 Algorithms, Spring 2016
Recitation #5 Worksheet

Streaming.

Sampling: Given a number k, you want to maintain a random sample of size k of the stream. I.e., 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 this prefix should be equally likely.

1. For $k = 1$, show that the following algorithm satisfies the required properties: 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.

   Solution: We claim that for any element from $e \in a_{[1..n]}$, we have e in our hand after step n w.p. $1/n$. The proof is inductive. The base case is easy. Consider the case for element e at some time n.

   - If $e \in a_{[1..n-1]}$, then we have it at time $n-1$ w.p. $\frac{1}{n-1}$. Now at time n, the chance we don't discard it is $\frac{n-1}{n}$, so multiplying we get $\frac{1}{n}$.
   - If $e = a_n$, then the chance we picked it at time n is $1/n$.

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?)

   Solution: The algorithm: pick the n^{th} item with probability k/n, and drop a uniformly random item from the current set.

   Claim: for any k-sized set S from $e \in a_{[1..n]}$, we have S after step n w.p. $\frac{1}{\binom{n}{k}}$. The proof is again inductive. The base case for $n = k$ is easy. Consider the case for element e at some time $n > k$.

   - If $S \subseteq a_{[1..n-1]}$, then we have it at time $n-1$ w.p. $\frac{1}{\binom{n-1}{k}}$. Now at time n, the chance we don't destroy it is $\frac{n-k}{n}$, so multiplying we get $\frac{k!(n-1-k)!}{(n-1)!} \cdot \frac{n-k}{n} = \frac{1}{\binom{n}{k}}$.
   - If $a_n \in S$, then look at $S_i = (S \setminus \{a_n\}) \cup \{a_i\}$ where $a_i \notin S$. There are $n-k$ such sets S_i. For each of the $n-k$ sets, inductively the chance we have it at time $n-1$ is $\frac{1}{\binom{n-1}{k}}$, then we have to drop a_i (w.p. $1/k$), and add a_n (w.p. $1/n$). Overall we get

   $$(n-k) \times \frac{k!(n-1-k)!}{(n-1)!} \times \frac{1}{k} \times \frac{k}{n} = \frac{1}{\binom{n}{k}}.$$

   This idea is called *Reservoir Sampling*.

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?

Solution: For one missing number, you can store the sum of all numbers seen so far. Then finally subtract that from $\frac{n(n+1)}{2}$ to get the missing number. This requires $2\log_2(n) + O(1)$ bits. This can be reduced to $\lceil \log_2(n) \rceil$ by keeping the `XOR` of all of the numbers instead of the sum. For two, you can store, e.g., the sum, and the sum of their squares. Then you'll know $a+b$ and a^2+b^2, and can solve for the answer. This is $3\log_2(n) + O(1)$ bits. Can you do it in $2\log_2(n) + O(1)$, which would be closer to the information theoretic lower bound? You could also have stored the sum and product of the numbers seen, but that requires more space. The product of the numbers could be as large as $\Omega(n!)$ which requires $\Omega(n \log n)$ bits to store.

Heavy Hitters Estimation.

In the heavy-hitters estimation with deletions (which is often called `Count-Min`[1]), we created an estimator with "one-sided error": our estimate was always an *over*estimate. I.e., for the target value v we created a random variable X such that $\mathbf{Pr}[X \geq v] = 1$. Suppose $\mathbf{E}[X] = \mu$.

1. Show that $\mathbf{Pr}[X \geq 2\mu] \leq \frac{1}{2}$.

   Solution: Markov's inequality.

2. Use this to show that if we take k independent copies $X_1, X_2, \ldots, X_k$ of the r.v. X, then $\mathbf{Pr}[min_{i=1}^{k}(X_i) \geq 2\mu] \leq 2^{-k}$.

   Solution: In order to get such a high estimate, we must get unlucky all k times. The prob of that is 2^{-k}.

3. Show that $k = \lg(1/\delta)$ gives $2^{-k} = \delta$.

   Solution: $2^{-k} = 2^{-\lg 1/\delta} = \delta$.

Fingerprinting.

Many Patterns: You are given a set of patterns $P_1, P_2, \ldots, P_k$ of equal length (all of them having length n) and a text T of length m. Give an algorithm to find all *good* locations in T, where a location i is good if some pattern among $P_1, P_2, \ldots P_k$ occurs as a substring of T starting at location i. The expected runtime should be $O(kn+m)$, and the probability of error is at most 0.01.[2]

[1] The original handout mistakenly said `MinHash`, which is a different algorithm

[2] Assume you can do arithmetic operations on numbers of size $O(\log(kmn))$ in constant time, even modulo a prime.

Solution: (Sketch.) Use Karp-Rabin fingerprinting and hashing. First, pick a random prime p in some set $[M]$ and compute Karp-Rabin fingerprints $g_p(P_j) = P_j \mod p$ of the P_js in time $O(kn)$. Store these fingerprints in a hash table of size k whose hash function h is chosen from a universal hash family. At each location i of the text, compute $g_p(T_{i\ldots i+(n-1)})$ in $O(1)$ time, hash this via h and look for matches over all the fingerprint values mapped to this location. If there is a match, then declare that that location in T to be good. In expectation there will be $O(1)$ fingerprints in that bucket of the hash table (since k fingerprints are being hashed into k locations), so the expected time for this is $O(1)$, and for the whole algorithm is $O(m+kn)$. The error probability is k times that in lecture, so choosing $M = \Theta(kmn \log(kmn))$ suffices.