CS 498ABD Spring 2019 — Midterm Solutions

Problem 1 Let $h : [n] \to [m]$ be a random hash function chosen from a 3-wise independent family of hash functions. For a fixed item i let Y be the number of items $i' \neq i$ that collide with i under h.

- What is $\mathrm{E}[Y]$?
- What is $\mathrm{Var}[Y]$ as a function of m, n? *Hint:* Use 3-wise independence here.
- Using Chebyshev, what is $\mathrm{P}[Y \geq a]$ where $a \geq 1$ is some integer. Express this as a function of a, m, n.

Solution:

- Let X_j be the indicator random varialbe for the event that $h(j) = h(i)$. Since h is pairwise independent, $\mathrm{E}[X_i] = \mathrm{P}[X_j = 1] = \frac{1}{m}$. Then $Y = \sum_{j \neq i} X_i$, so $\mathrm{E}[Y] = \boxed{\frac{n-1}{m}}$.
- Since h is 3-wise independent, the variables X_j are pairwise independent even after fixing i. Then since the X_i's are indicator random variables,

$$\mathrm{Var}\, Y = \sum_{j \neq i} \mathrm{Var}[X_i] = \sum_{j \neq i} (\mathrm{E}[X_i] - \mathrm{E}[X_i]^2) = \boxed{(n-1)\left(\frac{1}{m} - \frac{1}{m^2}\right)}.$$

- Using Chebyshev, we get

$$\begin{aligned} \mathrm{P}[Y \geq a] &= \mathrm{P}[|Y - \mathrm{E}[Y]| \geq a - \mathrm{E}[Y]] \\ &\leq \mathrm{Var}[Y]/(a - \mathrm{E}[Y])^2 \\ &= \boxed{\frac{n-1}{(a-(n-1)/m)^2} \cdot \left(\frac{1}{m} - \frac{1}{m^2}\right)} \end{aligned}$$

■

Problem 2 We have seen the use of the median trick for improving the probability of success. Suppose we have an estimator X for a quantity α of interest such that $\mathrm{E}[X] = \alpha$ and $\mathrm{P}[|X-\alpha| \geq \epsilon\alpha] < \rho$ for some $\rho < 1/2$. We wish to improve the error probability to δ for some desired δ. We have seen the use of the median trick for this. We compute independent estimators $X_1, X_2, \ldots, X_h$ in parallel and output the median Z of the computed estimators. How large should h be to guarantee that $\mathrm{P}[|Z-\alpha| \geq \epsilon\alpha] \leq \delta$ (as a function of ρ and δ)? Use one of the Chernoff inequalities and briefly justify your bound.

Solution: So set Y_i be the indicator random variable for the event that $|X-\alpha| \geq \epsilon\alpha$, and set $Y = \sum_{i=1}^{n} Y_i$. Recall that $|Z-\alpha| \geq \epsilon\alpha$ only if $Y \geq \frac{h}{2}$.

$\mathrm{E}[Y_i] < \rho$, so $\mathrm{E}[Y] < h\rho$. Then since $\frac{h}{2} = h\rho(1 + (\frac{1}{2\rho} - 1))$, setting $\lambda = \frac{1}{2\rho} - 1$ and $\mu = h\rho$,

$$\mathrm{P}\left[Y \geq \frac{h}{2}\right] \leq \mathrm{P}[Y \geq (1+\lambda)\mu] \leq \left(\frac{\mathrm{e}^{\lambda}}{(1+\lambda)^{1+\lambda}}\right)^{\mu}$$

which is at most δ if $\boxed{h \geq \dfrac{\ln \frac{1}{\delta}}{\rho - \ln \sqrt{2\mathrm{e}\rho}}}$. ■

Problem 3 Let $A[1..n]$ be a sorted array of n integers. Given an integer x, one way to decide if $x \in A$ is to use binary search. In this problem, we analyze a randomized version of binary search to find x.

Consider a randomized variant of binary search where one picks a *random* index $i \in [n]$ and compares $A[i]$ with x. If $A[i] = x$, then it terminates with the answer "yes"; if $A[i] \neq x$, then it recurses appropriately.

- Write down a formal description of randomized binary search including taking care of base cases.
- Prove that the expected running time for searching any given item x is $O(\log n)$.
- **Extra credit:** Prove that the running time of the algorithm is $O(\log n)$ with high probability.

Solution:

- The algorithm is as follows:

```
RandBinSearch(A[1..n], x)
if |A| = 0:
   return "not found"
pick i uniformly at random in [1, n]
if A[i] < x:
   return RandBinSearch(A[1..i-1], x)
else if A[i] > x:
   return RandBinSearch(A[i+1..n], x)
else:       // A[i] = x
   return "found"
```

- Let $T(n)$ be the worst-case expected number of comparisons on an array of size $\leq n$. Let X_i be the indicator random variable for the event that i as the pivot index. Then $T(n)$ satisfies the recurrence

$$T(n) \leq 1 + \sum_{i=1}^{n} \mathrm{P}[X_i = 1] \max\{T(i-1), T(n-i-1)\} \leq 1 + \frac{2}{n} \sum_{i=n/2+1}^{n} T(i-1).$$

We will show that $T(n) \leq 1 + c \lg i$ for some $c > 0$. The base case is $n = 1$: after 1 comparison, the algorithm either terminates (`found`) or recurses once and terminates immediately (`not found`). Otherwise,

$$\begin{aligned}
T(n) &\leq 1 + \frac{2}{n} \sum_{i=n/2+1}^{n} (1 + c \lg i) \\
&\leq 2 + \frac{2}{n} \sum_{i=n/2+1}^{3n/4} c \lg \frac{3n}{4} + \frac{2}{n} \sum_{i=3n/4+1}^{n} c \lg n \\
&= 2 - \frac{1}{2} c \lg \frac{4}{3} + c \lg n \\
&\leq c \lg n
\end{aligned}$$

for $c \geq 4/\lg \frac{4}{3} \approx 10$.

Alternatively: We can view the choices of pivots as follows. Generate a random permutation σ of $[n]$. At each round, we take the next available value in σ to be the pivot index. When we recurse, we remove all values that are either larger than or smaller than the current index, as appropriate.

Let X_i be the indicator random variable for the event that the $A[i]$ is chosen as a pivot while looking for x. There are three cases.

First suppose $A[i] < x$ and let k be the largest index such that $A[k] < x$. Then $A[i]$ is chosen as a pivot if and only if i comes before any of $i+1\,..\,k$ in σ. Since each of $[i,k]$ these has equal probability of being picked first, the probability of this happening is $\frac{1}{k-i+1}$.

Next, suppose $A[i] > x$ and let ℓ be the smallest index such that $A[\ell] > x$. Then $A[i]$ is chosen as a pivot if and only if i comes before any of $\ell\,..,i-1$ in σ. This happens with probability $\frac{1}{i-\ell+1}$.

Finally, if $A[i] = x$ then the probability of being chosen is 1.

In summary, the running time is bounded above by

$$1 + \sum_{i:A[i]<x} \frac{1}{k-i+1} + \sum_{i:A[i]>x} \frac{1}{i-\ell+1} \leq 1 + 2\sum_{i=1}^{n} \frac{1}{i} \leq 1 + 2H_n = O(\log n).$$

Alternatively: As seen below, the algorithm runs in $O(\log n)$ time with probability at least $1 - \frac{1}{n^4}$. On the other hand, the running time of the algorithm cannot exceed $O(n)$, since we can only pick n pivots in the worst case. So the expected running time is upper bounded by

$$\left(1 - \frac{1}{n^4}\right) O(\log n) + \frac{1}{n^4} O(n) = O(\log n).$$

- For each j, let S_j be the part of the array considered in the j-th level of recursion. Call the pivot picked in the j-th round *lucky* if its index is between $\frac{1}{4}|S_j|$ and $\frac{3}{4}|S_j|$. The probability of this happening is $\frac{1}{2}$.

  We claim that if the pivot is lucky, $|S_{j+1}| \leq \frac{3}{4}|S_j|$. If x is smaller than the pivot, the index of the pivot is at most $\frac{3}{4}|S_j|$ and we recurse left. If x is larger, the index of the pivot is at least $\frac{1}{4}|S_j|$ and we recurse right. In either case, $|S_{j+1}| \leq \frac{3}{4}|S_j|$.

  To get the base case, $4\ln n$ lucky pivot choices suffices. Just as in the lecture/homework, a Chernoff bound gives us that the probability of not getting $4\ln n$ lucky pivots in $32\ln n$ is at most $\frac{1}{n^4}$.

  Thus with probability $1 - \frac{1}{n^4}$, the algorithm finishes in $O(\log n)$ time. ■

Problem 4 Recall the algorithm to estimate the number of distinct elements in a stream using an ideal hash function $h : [n] \to [0,1]$. The algorithm maintains the minimum of the hash value seen in the stream, say z, and outputs $\frac{1}{z} - 1$ as the estimator for the number of distinct elements. Suppose there was a mistake in the implementation and instead of storing the minimum hash value seen, z stored the *maximum* hash value. How would you use z now to estimate the number of distinct elements? Briefly justify your answer.

Solution: Let $g : [n] \to [0,1]$ be defined by $g(x) = 1 - h(x)$. Then g is also an ideal hash function, and $1 - z$ is the minimum hash value under g. So we can return $\dfrac{1}{1-z} - 1 = \boxed{\dfrac{z}{1-z}}$. ■

Problem 5 Consider F_2 estimation via the AMS algorithm using 4-wise independent hash functions. In this problem, the high-level goal is to process two different streams coming in at two different locations and use this estimator to estimate the F_2 distance between the streams.

Let σ_1 and σ_2 be two streams. Let $\{f_{1,i} : i \in [n]\}$ and $\{f_{2,i} : i \in [n]\}$ denote the frequencies of σ_1 and σ_2, respectively. The F_2 distance between the streams is the sum

$$\sum_{i=1}^{n}(f_{1,i} - f_{2,i})^2.$$

Recall that the AMS estimator computes a value Z where the expected value of Z^2 is the F_2 of the stream. (One then takes averages and then medians of many copies to improve the accuracy.) The basic framework to estimate the F_2 distance of σ_1 and σ_2 is as follows. We first produce an estimate Z_1 for σ_1 and an estimate Z_2 for σ_2. We then somehow combine Z_1 and Z_2 to estimate the distance. Here we have two design decisions.

- When producing Z_1 and Z_2, should we use the same hash function, or two independent ones?
- How do we combine Z_1 and Z_2 so that the expected value is $\sum_{i=1}^{n}(f_{1,i} - f_{2,i})^2$.

Answer the above with some brief justification.

Solution: Recall that the AMS sketch supports deletions via subtraction and works for negative frequencies. We can interpret the F_2 distance between σ_1 and σ_2 as the F_2 of a single stream consisting of inserting the elements in σ_1 and then deleting the elements in σ_2. Thus we should $\boxed{\text{use the same hash function}}$, and output $\boxed{(Z_1 - Z_2)^2}$. ■