15-451/651 Algorithms, Spring 2019

Homework #3 **Due: February 22, 2019**

(25 pts) 1. **(Some Streaming Specialities)**

Suppose we are given a positive integer K, and we want to estimate the number of *distinct* elements in a stream $a_1, a_2, \ldots, a_n$. In particular, we want to find out if the number of distinct elements is at most K ("low"), or at least $2K$ ("high"). Here is an algorithm:

- Fix a 2-universal hash family $H : U \to \{0, 1, 2, \ldots, 4K-1\}$, and pick a hash function h from it.
- If for some $t \in \{1, 2, \ldots, n\}$, we have $h(a_t) = 0$, say "High"; else say "Low". (So, we don't need to keep a hash table to run this algorithm, just a flag that tells us if anything has hashed to 0.)

(i) Show that if the number of distinct elements is at most K, then

$$\mathbf{Pr}[\text{we say "High"}] \leq 1/4.$$

Solution: since H is 2-universal, it is also 1-universal, meaning that any given item $x \in \Sigma$ has probability $1/(4K)$ of hashing to any particular element, and in particular $\mathbf{Pr}[h(x) = 0] = 1/4K$. If "$A_i$" is the event that the i^{th} arriving element maps to 0. Then the probability there exists an element that hashes to 0 is $\mathbf{Pr}[\cup A_i] \leq \sum_i \mathbf{Pr}[A_i]$, the sum of the individual probabilities, which is at most $K \cdot (1/4K) = 1/4$.

(ii) Show that if the number of distinct elements is at least $2K$, then

$$\mathbf{Pr}[\text{we say "High"}] \geq 3/8.$$

(Hint: Show this for the case when the number of distinct elements is $2K$. How does the probability behave when this number increases.)

Solution: We use the Boole-Bonferroni inequalities, with "A_i" denoting the event that element i hashes to 0. Because H is 2-universal, we have that for all $i \neq j$, $\mathbf{Pr}(A_i \cap A_j) = 1/(4K)^2$. So, considering the first $2K$ elements, we have $\mathbf{Pr}[\cup_i A_i] \geq 2K/(4K) - ((2K)(2K-1)/2)(1/(4K)^2) \geq 1/2 - 1/8 = 3/8$.

(iii) Now to boost the success probability, we consider m copies of the above data structure, with independently chosen hash functions. Some of these might say "High" and some say "Low". If the fraction of Highs is larger than $\frac{1}{2}(\frac{1}{4} + \frac{3}{8}) = \frac{5}{16}$ then we say "High", else we say "Low". Prove that the probability that this boosted data structure makes a mistake is at most $2e^{-m/128}$. (Let's call this data structure HiLo(K).)

Solution: We apply Hoeffding's inequality where the independent random variables correspond to saying High or Low in each independent copy of the data structure.

Namely, let $X_j = 1$ if the j^{th} copy of the data structure returns High, else let $X_j = 0$. If there are indeed fewer than K elements, the expected value of $\sum_{i=1}^{m} X_i$ is at most $m/4$ (by linearity of expectations), whereas if there are more than $2K$ elements, the expected value is at least $3m/8$. In either case, the probability we make a mistake is at most the probability $\frac{1}{m}\sum_i X_i$ differs from the expected fraction of Highs by $\varepsilon = 1/16$, which by Hoeffding's inequality is at most $2e^{-2m/16^2} = 2e^{-m/128}$.

(iv) Run one copy of HiLo(K) for each value of $K = 1, 2, 4, 8, \ldots, U/2$. As an estimate for $\|x\|_0$, output the *smallest* value K^* for which HiLo(K^*) says "Low". Show that if $m = c(\ln \ln U + \log 1/\eta)$ for some large enough constant c, all copies of HiLo will answer correctly with probability at least $1 - \eta$. In this case observe that our estimate is correct to within a factor of 4. Also, ignoring the space to represent your hash function, how many bits of space do you need to store your entire data structure? Try to optimize this.

Solution: (Sketch.) Each copy is correct with probability $1 - 2e^{-m/128}$, so if we set $m = 128(\ln(\frac{2}{\eta} \ln U))$ then the $2e^{-m/128} = \frac{\eta}{\ln U}$. And now taking a union bound over all $\ln U$ copies of HiLo means that the probability that any of the copies will make a mistake is at most η. This proves the first part. Now notice that you just need to keep track of the smallest index K^*: since there are $\log U$ possible indices, you need only $O(\log \log n)$ bits.

One can get a $(1 + O(\varepsilon))$-approximate estimate by looking at powers of $(1 + \varepsilon)$ instead of powers of 2, we just kept it to powers of 2 to make it cleaner.

Hints: For the first two parts, recall the Boole-Bonferroni inequalities which state that for any set of events A_i we have

$$\sum_i \mathbf{Pr}(A_i) - \sum_{i<j} \mathbf{Pr}(A_i \cap A_j) \leq \mathbf{Pr}\left[\bigcup_i A_i\right] \leq \sum_i \mathbf{Pr}(A_i).$$

For the third, use Hoeffding's inequality, which states that for m independent 0-1 valued random variables $X_1, \ldots, X_m$ with $p_i = \mathbf{E}[X_i]$, we have

$$\mathbf{Pr}\left[\left|\frac{1}{m}\sum_{i=1}^{m} X_i - \frac{1}{m}\sum_{i=1}^{m} p_i\right| \geq \epsilon\right] \leq 2e^{-2m\epsilon^2}.$$

(25 pts) 2. **(Rolle thee Olde Diee)**

Roll-the-Die is played by two players A and B. The board has positions $0, 1, \ldots, n$. The game state is defined by the position of A and the position of B on the board. The players take turns rolling a standard 6-sided die and moving toward zero by that many steps. The first player to reach (or pass) position 0 wins.

(a) Give an $O(n^2)$ algorithm that computes the probability P_{ij} that the first player wins, given that the first player starts at i and the second starts at j, with $i, j \in \{0, 1, \ldots, n\}$.

(b) Now change the problem slightly: if a 6 is rolled, then the player does not move. Solve part (a) with these changed rules.

Solution: (a) Let F_{ij} be the probability that the *first* person wins if the first person is at i and the second at j. The base case is $F_{ij} = 1$ for $i \leq 0, j \geq 1$, and $F_{ij} = 0$ for $i \geq 1, j \leq 0$. The position is undefined for F_{00}. Now we give a recurrence for F_{ij} for $i, j \geq 1$. Now we get the recurrence that

$$F_{i,j} = \frac{1}{6}\sum_{d=1}^{6}(1 - F_{j,i-d}) = 1 - \sum_{d=1}^{6}\frac{F_{j,i-d}}{6}. \tag{1}$$

Since $i - d < i$, we can compute the values of $F_{a,b}$ in increasing order of $a + b$.

(b) Let F_{ij} be the probability that the *first* person wins if the first person is at i and the second at j. The base case is $F_{ij} = 1$ for $i \leq 0, j \geq 1$, and $F_{ij} = 0$ for $i \geq 1, j \leq 0$. The position is undefined for F_{00}. Now we give a recurrence for F_{ij} for $i, j \geq 1$.

For the rest of the analysis, imagine that the die gives values $0, 1, 2, 3, 4, 5$. Now we get the recurrence that

$$F_{i,j} = \frac{1}{6}\sum_{d=0}^{5}(1 - F_{j,i-d}) = 1 - \frac{F_{j,i}}{6} - \sum_{d=1}^{5}\frac{F_{j,i-d}}{6}. \tag{2}$$

We cannot just use this recurrence, because $F_{i,j}$ depends on $F_{j,i}$, which in turn depends on $F_{i,j}$. So we'll get an infinite loop this way. Let's see what we get writing the same recurrence for $F_{j,i}$.

$$F_{j,i} = \frac{1}{6}\sum_{d=0}^{5}(1 - F_{i,j-d}) = 1 - \frac{F_{i,j}}{6} - \sum_{d=1}^{5}\frac{F_{i,j-d}}{6}. \tag{3}$$

$$= 1 - \frac{1}{6}\left(1 - \frac{F_{j,i}}{6} - \sum_{d=1}^{5}\frac{F_{j,i-d}}{6}\right) - \sum_{d=1}^{5}\frac{F_{i,j-d}}{6} \qquad \text{(plugging in (2))} \tag{4}$$

$$= \frac{5}{6} + \frac{F_{j,i}}{36} + \sum_{d=1}^{5}\frac{F_{j,i-d} - 6F_{i,j-d}}{36} \tag{5}$$

Moving $F_{j,i}$ over and simplifying,

$$= \frac{36}{35}\left(\frac{5}{6} + \sum_{d=1}^{5}\frac{F_{j,i-d} - 6F_{i,j-d}}{36}\right) = \frac{6}{7} + \sum_{d=1}^{5}\frac{F_{j,i-d} - 6F_{i,j-d}}{35}. \tag{6}$$

(25 pts) 3. **(Palindrome Deconstruction)**

As you know a palindrome is a string which is invariant under reversal. For example "madamimadam" is a palindrome. As are "x" and "yy" and "xyyx" and "pqp", or indeed, the empty string.

Given a string S you can reduce it to nothing by repeatedly deleting palindromic substrings. For example "xabayx" can be "deconstructed" by removing the palindrome "aba" leaving "xyx". Now we finish the job by removing "xyx". So two palindrome removals are sufficient. You could also do it in six operations by removing one character at a time. (A single character is always a palindrome.)

Give an $O(n^3)$ algorithm to compute the minimum number of palindrome removals necessary to deconstruct a string S of length n.

Solution: For string S, let $PD(S)$ denote the optimal number of palindrome removals. The length of a palindrome removal sequence is the number of palindromes removed in it.

Claim: For a string $S = xTx$ and $T \neq \emptyset$, then $PD(S) \leq PD(T)$.

Proof: $PD(S) \leq PD(T)$, because take a palindrome destruction sequence for T. Whichever is the last palindrome to be removed, add an x to either end of it. This is also a palindrome, and now you have a sequence for S of the same length. ■

OK, back to business. Let $S[1], \ldots, S[n]$ denote the n positions of the S. We will maintain a table $T(i, j)$ containing the value $PD(S[i \ldots j])$.

- $S[i, i] = 1$ for all i.
- $S[i, i+1] = 2$ if $S[i] \neq S[i+1]$ and 1 if $S[i] = S[i+1]$.
- For $j \geq i + 2$, define $temp[i, j] = \min_{k=i \ldots j-1}(S[i, k] + S[k+1, j])$. Then

$$S[i, j] = \begin{cases} \min(S[i+1, j-1], temp[i, j]) & \text{if } S[i] = S[j] \\ temp[i, j] & \text{otherwise} \end{cases}$$

Basically, the place where the first element x is matched will split the string in two, and the best way to do this is stored in $temp$. Or else x will be matched to the x at the other end, and now we can use the Claim above.