Advanced Algorithms 2024 28/10, 2024

Sample Solutions 06

Lecturer: Maximilian Probst *Teaching Assistant: Patryk Morawski*

1 Reducing the Variance

Since X is an unbiased estimator, and we know its variance, we would like to apply Chebyshev's inequality to bound the probability that X is far away from its expectation. Unfortunately, the variance from a single run of the algorithm is too large for Chebyshev to give us our desired bound. Luckily though, we can easily fix this by taking the mean over multiple runs of the algorithm - which will decrease the variance linearly in the number of runs.

More formally, for some $k \in \mathbb{N}$, we run our algorithm $\mathcal{A}$ independently k-times and let $X_1, \ldots, X_k$ be the estimates we get from each of these runs. Moreover, we note that the X_i's are independent and let $Y = \frac{\sum_{i=1}^{k} X_k}{k}$. We have $\mathbb{E}[Y] = x$.

Let us now compute the variance of Y, i.e.,

$$\mathrm{Var}[Y] = \mathrm{Var}[\frac{\sum_{i=1}^{k} X_i}{k}] = \frac{1}{k^2}\sum_{i=1}^{k}\mathrm{Var}[X_i] = \frac{1}{k}\mathrm{Var}[X],$$

where we used that $\mathrm{Var}[a(X+Y)] = a^2(\mathrm{Var}[X] + \mathrm{Var}[Y])$ for any $a \in \mathbb{R}$ and any independent random variables X and Y.

Now, by Chebyshev's inequality we get

$$\Pr[|Y - x| \geq 0.001x] = \Pr[|Y - \mathbb{E}[Y]| \geq 0.001x] \leq \frac{10^6 x^2}{k \cdot 10^{-6} x^2} \leq 0.01,$$

for $k \geq 10^{14}$. Thus the algorithm obtained by running $\mathcal{A}$ at least 10^{14} and taking the mean over the single estimates satisfies the conditions of the task.

2 Morris's Approximate Counting Algorithm

1. Similarly to the computation of $\mathbb{E}[2^{Y_m}]$ we have seen in the lecture, we want to proceed by induction on m. On the first element, i.e., $m = 1$, Y increments to 1 with probability 1, so $\mathcal{E}[2^{2Y_m}] = 4 = \frac{3}{2}m^2 + \frac{3}{2}m + 1$ holds. Now, suppose that for some $m \in \mathbb{N}^+$ we have proved the statement. Then,

$$\begin{aligned}
\mathbb{E}[2^{2Y_{m+1}}] &= \sum_{j=1}^{m} \mathbb{E}[2^{2Y_{m+1}} | Y_m = j] \Pr[Y_m = j] && \text{Condition on } Y_m \\
&= \sum_{j=1}^{m} (2^{2(j+1)} \cdot 2^{-j} + 2^{2j} \cdot (1 - 2^{-j})) \cdot \Pr[Y_m = j] && \text{We increment } Y_{m+1} \text{ w.p. } 2^{-Y_m} \\
&= \sum_{j=1}^{m} (4 \cdot 2^j + 2^{2j} - 2^j) \cdot \Pr[Y_m = j] && \text{Simplifying} \\
&= 3(\sum_{j=1} 2^j \cdot \Pr[Y_m = j]) + (\sum_{j=1}^{m} 2^{2j} \cdot \Pr[Y_m = j]) && \text{Reordering}
\end{aligned}$$

$$
\begin{aligned}
&= 3\mathbb{E}[2^{Y_m}] + \mathbb{E}[2^{2Y_m}] && \text{Definition of } \mathbb{E}[2^{Y_m}] \text{ and } \mathbb{E}[2^{2Y_m}] \\
&= 3(m+1) + \frac{3}{2}m^2 + \frac{3}{2}m + 1 && \text{Induction + Lecture} \\
&= \frac{3}{2}(m+1)^2 + \frac{3}{2}(m+1) + 1 && :)
\end{aligned}
$$

2. By the definition of the variance

$$
\begin{aligned}
\mathrm{Var}[2^{Y_m} - 1] &= \mathbb{E}[(2^{Y_m} - 1 - (m+1))^2] \\
&= \mathbb{E}[2^{2Y_m} - 2m \cdot 2^{Y_m} + m^2] \\
&= \frac{3}{2}m^2 + \frac{3}{2}m + 1 - 2m \cdot (m+1) + m^2 \\
&= \frac{m^2}{2} - \frac{m}{2} + 1 \leq \frac{m^2}{2},
\end{aligned}
$$

for $m \geq 2$. For $m = 1$, notice that Y_m is deterministically 1, so $\mathrm{Var}[2^{Y_m} - 1] = 0 \leq \frac{1}{2}$.

3 Bounding the Probability of a Collision in FM+

1. For distinct $i, j \in S$ let $X_{i,j}$ be the indicator random variable for the event that $g(i) = g(j)$ and let $X = \sum_{\substack{i,j \in [b] \\ i \neq j}} X_{i,j}$ be a random variable counting the number of pairwise collisions of elements of S. Then, $\mathbb{E}[X_{i,j} = \frac{1}{C\epsilon^{-4}\log^2 n}$ and therefore

$$
\mathbb{E}[X] = \binom{|S|}{2} \frac{1}{C\epsilon^{-4}\log^2 n} \leq \frac{|S|^2}{C\epsilon^{-4}\log^2 n} \leq \frac{C'^2\epsilon^{-4}(\log n + 2)^2}{C\epsilon^{-4}\log^2 n} \leq \frac{1}{6},
$$

for $C \geq 9C'^2$.

Note that there exist some distinct $i, j \in S$ with $g(i) = g(j)$ if and only if $X \geq 1$. Using Markov's inequality we therefore get

$$
\Pr[\exists_{\substack{i,j \in S \\ i \neq j}} . g(i) = g(j)] = \Pr[X \geq 1] \leq \frac{1}{6}.
$$

2. Note that under this assumption, any time we add execute the line $\mathcal{B} \geq \mathcal{B} \cup \{(g(a_i), ZEROS(h(a_i))\}$ for some $a_i = j$ such that j hasn't appeared in $\mathcal{B}$ yet, we increase the size of $\mathcal{B}$ by one. In particular, between any two increments of X, we can add at most $C' \cdot \epsilon^{-2}$ new $i \in [n]$ to $\mathcal{B}$.

   Moreover, since $ZEROS(h(i)) \leq n$ for each $i \in [n]$, we have $X \leq \log n + 1$ at the end of the execution. This means, that we increment X at most $\log n + 2$ times during the whole algorithm. Together with the above observation this implies that $|S| \leq C' \cdot \epsilon^{-2}(\log n + 2)$.

3. Let the input stream $a_1, \ldots, a_m$ be given. First, consider a modified FM+ algorithm $\mathcal{A}'$, where we replace the line $\mathcal{B} \leftarrow \mathcal{B} \cup \{(g(a_i), ZEROS(h(a_i)))\}$ with $\mathcal{B} \leftarrow \mathcal{B} \cup \{a_i, ZEROS(h(a_i)))\}$. That is, $\mathcal{A}$'s simulates the run of the FM+ algorithm where there are no collisions under our hash function g. For a given stream input, let S' be the set of elements that appeared in $\mathcal{B}$ during the execution of $\mathcal{A}'$. Then, by subtask 2, we have that $|S'| \leq C' \cdot \epsilon^{-2}(\log n + 2)$.

   Now, consider a run of the unmodified FM+ algorithm on this input sequence. By subtask 1, the event E that for any distinct $i, j \in S'$ we have $g(i) \neq g(j)$ happens with probability at least 5/6, . Moreover, conditioned on the event E, our unmodified FM+ algorithm, will do exactly the same steps as $\mathcal{A}'$. In particular, the set S of elements that appeared in $\mathcal{B}$ during the execution will be exactly the same as S', i.e., for any distinct $i, j \in S = S'$ we have $g(i) \neq g(j)$. This shows that the probability of a collision under g of two elements that appeared in $\mathcal{B}$ during the execution of FM+ is at most 1/6.

4 Majority Element

1. We want to consider the following algorithm.

Algorithm 1 MajorityStream($S = \{a_1, \ldots, a_m\}$)

```
guess ← 0
count ← 0
for a_i ∈ S do                          ▷ Items arrive in streaming fashion
    if a_i = guess then
        count ← count + 1
    else if count > 1 then
        count ← count − 1
    else
        guess ← a_i
    end if
end for
return guess
```

Clearly, to keep track of the guess and the counter, we need at most $\mathcal{O}(\log m + \log n)$ space. Therefore, it remains to prove the correctness of the algorithm.

Let therefore $a_1, \ldots, a_m \in [n]$ be the input stream and suppose there is an element $j \in [n]$ s.t. $a_i = j$ for more than $m/2$ of the a_i's. We will analyze the algorithm step-by-step as the elements of the stream arrive. Let therefore, $guess_i$ and $counter_i$ be the values of the variables after we have processed the element a_i. Moreover, we let $n_i = |\{1 \leq i' \leq i : a_{i'} = j\}|$ denote the number of times the majority element j appears among the first i elements. We will show the following statement by induction on i.

Claim 1. *For each $i \in [m]$ the following holds. If $2n_i > i$, then $guess_i = j$ and $counter_i \geq 2n_i - i$. If $2n_i \leq i$ then either $guess_i = j$ or $counter_i \leq i - 2n_i + 1$.*

Indeed, the claim clearly holds after inserting the first element. Now suppose that the claim holds for some $i \in [n]$. We have the following possible cases:

(a) $2n_i > i$ and $a_{i+1} = j$. Then $guess_{i+1} = j$ and $counter_{i+1} = counter_i + 1 \geq 2(n_i + 1) - (i + 1)$.

(b) $2n_i > i$ and $a_{i+1} \neq j$. The updated counter for j, will be at least $2n_i - i - 1 = 2n_{i+1} - i - 1$. In case this value is more than 0, we are done. Otherwise, it we might get $guess_{i+1} = a_{i+1}$ and $counter_{i+1} = 1$, but then $2n_{i+1} \leq i + 1$, so this is fine.

(c) $2n_i \leq i$ and $a_{i+1} = j$. If $2n_{i+1} > i$ we have either $guess_i = j$ or $counter_i = 1$, so $counter_{i+1} \geq 1$ and $guess_{i+1} = j$. Otherwise, we either have $guess_{i+1} = j$ or $counter_{i+1} \leq i - 2n_i + 1 - 1 = (i + 1) - 2n_{i+1} + 1$.

(d) $2n_i \leq i$ and $a_{i+1} \neq j$. Then, either $guess_{i+1} = j$ or $counter_{i+1} \leq i - 2n_i + 1 + 1 = (i + 1) - 2n_i + 1$.

So the claim holds for $i + 1$ as well.

In particular, by the claim for $i = m$, since $2n_m > m$, we have that $guess_m = j$, as we were supposed to show.

2. Let $m = n$ and for simplicity suppose n is even. We can prove the claim by focusing on the streams of the form $S = \{a_1, a_2, \ldots, a_{n/2}, a_{n/2+1}, \ldots, a_n\}$, such that $a_1 < a_2 < \ldots < a_{n/2}$. So, the first half of S contains $\frac{n}{2}$ distinct elements in the increasing order. Note that

$\binom{n}{n/2} \geq \frac{2^n}{n+1}$ since $\sum_{i=0}^{n} \binom{n}{i} = 2^n$ and $\binom{n}{n/2} \geq \binom{n}{k}$ for every $0 \leq k \leq n$ (using Stirling's approximation, we can get a slightly better bound, say $\binom{n}{n/2} = \Theta(\frac{2^n}{\sqrt{n}})$). As a result, we have at least $\frac{2^n}{n+1}$ streams of length n where the first half of the stream is a sequence of elements in $[n]$ in strictly increasing order.

We prove the claim by contradiction. Suppose that we have a deterministic streaming algorithm for deciding the existence of majority using $n - (\log n + 1)$ bits of memory. Let A and B be two different sorted sequence of length $\frac{n}{2}$, each contains $\frac{n}{2}$ distinct elements from $[n]$. Run the algorithm on streams $S_1 = \{A, x, x, \ldots [\frac{n}{2} \text{ times}] \ldots, x\}$ and $S_2 = \{B, x, x, \ldots [\frac{n}{2} \text{ times}] \ldots, x\}$ where x is an element appeared in A but not in B. This element should exists as A and B are different and both of them has exactly $\frac{n}{2}$ distinct elements. Observe that a correct algorithm should return 1 for S_1 (since x appears more that $\frac{n}{2}$ times), and 0 for S_2 (since there is no element occurring more than $\frac{n}{2}$ times). Since the algorithm stores $n - (\log n + 1)$ bits, at most

$$2^{n-(\log n+1)} = \frac{2^n}{2n}$$

different configurations can be stored in the memory. So by pigeonhole principle, there are two A and B with the above properties such that content of the memory of the algorithm is same after it receives A or B. This forces the algorithm to output a similar value for S_1 and S_2 while the output for these streams should be different. This means that any correct algorithm for this problem has to store at least $n - \log n = \Omega(n)$ bits.