Chapter 24

Streaming and the Multipass Model

> I don't know why it should be, I am sure; but the sight of another man asleep in bed when I am up, maddens me. It seems to me so shocking to see the precious hours of a man's life - the priceless moments that will never come back to him again - being wasted in mere brutish sleep.
>
> Jerome K. Jerome, Three men in a boat

By Sariel Har-Peled, April 26, 2022[1]

24.1. The secretary problem

Assume that we are seeing n applications: $\alpha_1, \ldots, \alpha_n$, where the quality of each one of them is an independent random event. We would like to hire the best one, but we have to make an immediate decision – we see candidate α_i, we either hire them (and then we are stuck with them), or we let them go and see the next candidate. We win the game if we hire the best candidate out of the n candidate.

The question is what is the natural strategy to win the game? Let $P(r)$ be the probability that we win, when the strategy is to see the first $r-1$ candidates, and then hire the first candidate we see in $\alpha_r, \ldots, \alpha_n$ that is better than all the candidates seen in $\alpha_1, \ldots, \alpha_r$. We have that

$$\begin{aligned}
P(r) &= \sum_{i=1}^{n} \mathbb{P}\Big[\text{applicant } i \text{ is selected} \cap \text{applicant } i \text{ is the best}\Big] \\
&= \sum_{i=1}^{n} \mathbb{P}[\text{applicant } i \text{ is selected} \mid \text{applicant } i \text{ is the best}] \cdot \mathbb{P}[\text{applicant } i \text{ is the best}] \\
&= \left[\sum_{i=1}^{r-1} 0 + \sum_{i=r}^{n} P\left(\begin{array}{l}\text{the best of the first } i-1 \text{ applicants} \\ \text{is in the first } r-1 \text{ applicants}\end{array} \middle| \text{applicant } i \text{ is the best}\right)\right] \cdot \frac{1}{n} \\
&= \left[\sum_{i=r}^{n} \frac{r-1}{i-1}\right] \cdot \frac{1}{n} \\
&= \frac{r-1}{n} \sum_{i=r}^{n} \frac{1}{i-1}.
\end{aligned}$$

Observe that

$$\sum_{i=r}^{n} \frac{1}{i-1} \leq \int_{x=r-2}^{n-1} \frac{1}{x}\, \mathrm{d}x = \ln(n-2) - \ln(r-2) \approx \ln \frac{n}{r}.$$

For $r = n/e$, we have that $P(r) \approx \frac{r}{n} \ln \frac{n}{r} = \frac{1}{e} \ln e = 1/e$.

[1]

24.2. Reservoir sampling: Fishing a sample from a stream

Imagine that you are given a stream of elements $s_1, s_2, \ldots$, and you need to sample k numbers from this stream (say, without repetition) – assume that you do not know the length of the stream in advance, and furthermore, you have only $O(k)$ space available for you. How to do that efficiently?

There are two natural schemes:

(A) Whenever an element arrives, generate a random number for it in the range $[0, 1]$. Maintain a heap with the k elements with the lowest priority. Implemented naively this requires $O(\log k)$ comparisons after each insertion, but it is not difficult to improve this to $O(1)$ comparisons in the amortized sense per insertion. Clearly, the resulting set is the desired random sample

(B) Let S_t be the random sample maintained in the tth iteration. When the ith element arrives, the algorithm flip a coin that is heads with probability $\min(1, k/i)$. If the coin is heads then it inserts s_i to S_{i-1} to get S_i. If S_{i-1} already have k elements, then first randomly delete one of the elements.

Theorem 24.2.1. *Given a stream of elements, one can uniformly sample k elements (without repetition), from the stream using $O(k)$ space, where $O(1)$ time is spent for handling each incoming element.*

Proof: We implement the scheme (B) above. We only need to argue that this is a uniform random sample. The claim trivially hold for $i = k$. So assume the claim holds for $i < t$, and we need to prove that the set after getting tth element is still a uniform random sample.

So, consider a specific set $K \subseteq \{s_1, \ldots, s_t\}$ of k elements. The probability of K to be a random sample of size k from a set of t elements is $1/\binom{t}{k}$. We need to argue that this probability remains the same for this scheme.

So, if $s_t \notin K$, then we have

$$\mathbb{P}[K = S_t] = \mathbb{P}[K = S_{t-1} \text{ and } s_t \text{ was not inserted}] = \frac{1}{\binom{t-1}{k}}\left(1 - \frac{k}{t}\right) = \frac{k!(t-1-k)!(t-k)}{(t-1)!t} = \frac{1}{\binom{t}{k}}.$$

If $s_t \in K$, then

$$\mathbb{P}\big[K = S_t\big] = \mathbb{P}\left[\begin{array}{c} K \setminus \{s_t\} \subseteq S_{t-1}, \\ s_t \text{ was inserted} \\ \text{and } S_{t-1} \setminus K \text{ thrown out of } S_{t-1} \end{array}\right] = \frac{t-1-(k-1)}{\binom{t-1}{k}}\frac{\cancel{k}}{t}\frac{1}{\cancel{k}} = \frac{(t-k)k!(t-1-k)!}{(t-1)!t} = \frac{1}{\binom{t}{k}},$$

as desired. Indeed, there are $t - 1 - (k - 1)$ subsets of size k of $\{s_1, \ldots, s_{t-1}\}$ that contains $K \setminus \{s_t\}$ – since we fix $k - 1$ of the $t - 1$ elements. ■

24.3. Sampling and median selection revisited

Let $B[1, \ldots, n]$ be a set of n numbers. We would like to estimate the median, without computing it outright. A natural idea, would be to pick k elements $e_1, \ldots, e_k$ randomly from B, and return their median as the guess for the median of B.

In the following, let $B_{\langle t \rangle}$ be the tth smallest number in the array B.

Observation 24.3.1. *For any $\varepsilon \in (0, 1)$, we have that $\frac{1}{1-\varepsilon} \geq 1 + \varepsilon$.*

Lemma 24.3.2. *Let $\varepsilon \in (0, 1/2)$ be a fixed parameter, and let B be a set of n numbers. Let Z be the median of the random sample (with replacement) of B of size k. We have that*

$$\mathbb{P}\Big[B_{\langle \frac{1-\varepsilon}{2} n\rangle} \leq Z \leq B_{\langle \frac{1+\varepsilon}{2} n\rangle}\Big] \geq 1 - \delta, \qquad \textit{where} \qquad k \geq \left\lceil \frac{12}{\varepsilon^2} \ln \frac{2}{\delta} \right\rceil.$$

Namely, with probability at least $1 - \delta$, the returned value Z is $(\varepsilon/2)n$ positions away from the true median.

Proof: Let $L = B_{\langle (1-\varepsilon)n/2\rangle}$, and let e_i be the ith sample number, for $i = 1, \ldots, k$. Let $X_i = 1$ if and only if $e_i \leq L$. We have that

$$\mathbb{P}[X_i = 1] = \frac{(1-\varepsilon)n/2}{n} = \frac{1-\varepsilon}{2}.$$

As such, setting $Y = \sum_{i=1}^{k} X_i$, we have

$$\mu = \mathbb{E}[Y] = \frac{1-\varepsilon}{2} k \geq \frac{k}{4} \geq \frac{3}{\varepsilon^2} \ln \frac{2}{\delta}.$$

One case of failure of the algorithm is if $Y \geq k/2$. Since $\frac{1}{1-\varepsilon} \geq 1 + \varepsilon$, we have that

$$\mathbb{P}[Y \geq k/2] = \mathbb{P}\left[Y \geq \frac{1/2}{(1-\varepsilon)/2} \cdot \frac{1-\varepsilon}{2} k\right] \leq \mathbb{P}[Y \geq (1+\varepsilon)\mu] \leq \exp\left(-\frac{\varepsilon^2 \mu}{3}\right) \leq \exp\left(-\frac{\varepsilon^2}{3} \cdot \frac{3}{\varepsilon^2} \ln \frac{2}{\delta}\right) \leq \frac{\delta}{2}.$$

by Chernoff's inequality (see Lemma 24.6.1).

This implies that $\mathbb{P}\big[B_{\langle (1-\varepsilon)n/2\rangle} > Z\big] \leq \delta/2$.

The claim now follows by realizing that by symmetry (i.e., revering the order), we have that $\mathbb{P}\big[Z > B_{\langle (1+\varepsilon)n/2\rangle}\big] \leq \delta/2$, and putting these two inequalities together. ■

The above already implies that we can get a good estimate for the median. We need something somewhat stronger – we state it without proof since it follows by similarly mucking around with Chernoff's inequality.

Lemma 24.3.3. *Let $\varepsilon \in (0, 1/2)$, let B an array of n elements, and let $S = \{e_1, \ldots, e_k\}$ be a set of k samples picked uniformly and randomly from B. Then, for some absolute constant c, and an integer k, such that $k \geq \left\lceil \frac{c}{\varepsilon^2} \ln \frac{1}{\delta} \right\rceil$, we have that*

$$\mathbb{P}\big[S_{\langle k_-\rangle} \leq B_{\langle n/2\rangle} \leq S_{\langle k^+\rangle}\big] \geq 1 - \delta.$$

for $k_- = \lfloor (1-\varepsilon)k/2 \rfloor$, and $k^+ = \lfloor (1+\varepsilon)k/2 \rfloor$.

One can prove even a stronger statement:

$$\mathbb{P}\big[B_{\langle (1-2\varepsilon)n/2\rangle} \leq S_{\langle (1-\varepsilon)k/2\rangle} \leq B_{\langle n/2\rangle} \leq S_{\langle (1+\varepsilon)k/2\rangle} \leq B_{\langle (1+2\varepsilon)n/2\rangle}\big] \geq 1 - \delta$$

(the constant c would have to be slightly bigger).

24.3.1. A median selection with few comparisons

The above suggests a natural algorithm for computing the median (i.e., the element of rank $n/2$ in B). Pick a random sample S of $k = O(n^{2/3}\log n)$ elements. Next, sort S, and pick the elements L and R of ranks $(1-\varepsilon)k$ and $(1+\varepsilon)k$ in S, respectively. Next, scan the elements, and compare them to L and R, and keep only the elements that are between. In the end of this process, we have computed:

(A) α: The rank of the number L in the set B.

(B) $T = \{x \in B \mid L \le x \le H\}$.

Compute, by brute force (i.e., sorting) the element of rank $n/2-\alpha$ in T. Return it as the desired median. If $n/2-\alpha$ is negative, then the algorithm failed, and it tries again.

Lemma 24.3.4. *The above algorithm performs $2n+O(n^{2/3}\log n)$ comparisons, and reports the median. This holds with high probability.*

Proof: Set $\varepsilon = 1/n^{1/3}$, and $\delta = 1/n^{O(1)}$, and observe that Lemma 24.3.3 implies that with probability $\ge 1-1/\delta$, we have that the desired median is between L and H. In addition, Lemma 24.3.3 also implies that $|T| \le 4\varepsilon n \le 4n^{2/3}$, which readily implies the correctness of the algorithm.

As for the bound on the number of comparisons, we have, with high probability, that the number of comparisons is

$$O(|S|\log|S| + |T|\log|T|) + 2n = O\left(\sqrt{n}\log^2 n + n^{2/3}\log n\right) + 2n,$$

since deciding if an element is between L and H requires two comparisons. ■

Lemma 24.3.5. *The above algorithm can be modified to perform $(3/2)n + O(n^{2/3}\log n)$ comparisons, and reports the median correctly. This holds with high probability.*

Proof: The trick is to randomly compare each element either first to L or first to H with equal probability. For elements that are either smaller than L or bigger than H, this requires $(3/2)n$ comparisons in expectation. Thus improving the bound from $2n$ to $(3/2)n$. ■

Lemma 24.3.6. *Consider a stream B of n numbers, and assume we can make two passes over the data. Then, one can compute exactly the median of B using:*

(I) $O(n^{2/3})$ space.

(II) $1.5n + O(n^{2/3}\log n)$ comparisons.

The algorithm reports the median correctly, and it succeeds with high probability.

Proof: Implement the above algorithm, using the random sampling from Theorem 24.2.1. ■

Remark 24.3.7. Interestingly, one can do better if one is more careful. The basic idea is to do thinning – given two sorted sequence of sizes s, consider merging the sets, and then picking all the even rank elements into a new sequence. Clearly, the element of rank i in the output sequence, has rank $2i$ in the union of the two original sequences. A sequence that is the result of i such rounds of thinning is of ***level*** i. We maintain $O(\log n)$ such sequences as we read the stream. At any time, we have two buffers of size s, that we fill up from the stream. Whenever the two buffers fill up completely, we perform the thinning operation on them, creating a sequence of level 1.

If during this process we have two sequences of the same level, we merge them and perform thinning on them. As such, we maintain $O(\log n)$ buffers sequences each of size s. Assume that our stream has size n, and n is a power for 2. Then in the end of process, we would have only a single sequence of level

$h = \log_2(n/s)$. By induction, it is easy to prove that an element of rank r in this sequence, has rank between $2^h(r-1)$ and $2^h r$ in the original stream.

Thus, setting $s = \sqrt{n}$, we get that after a single pass, using $O(\sqrt{n} \log n)$ space, we have a sorted sequence, where the rank of the elements is roughly $\sqrt{n}$ approximation to the true rank. We pick the two consecutive elements (or more carefully, the predecessor, and successor), and filter the stream again, keeping only the elements in between these two elements. It is to show that $O(\sqrt{n})$ would be kept, and we can extract the median using $O(\sqrt{n} \log n)$ time.

We thus got that one can compute the median in two passes using $O(\sqrt{n} \log n)$ space. It is not hard to extend this algorithm to α-passes, where the space required becomes $O(n^{1/\alpha} \log n)$.

This elegant algorithm goes back to 1980, and it is by Munro and Paterson [MP80].

24.4. Big data and the streaming model

Here, we are interested in doing some computational tasks when the amount of data we have to handle is quite large (think terabytes or larger). The main challenge in many of these cases is that even reading the data once is expensive. Running times of $O(n \log n)$ might not be acceptable. Furthermore, in many cases, we can *not* load all the data into memory.

In the ***streaming*** model, one reads the data as it comes in, but one can not afford to keep all the data. A natural example would be a internet router, which has gazillion of packets going through it every minute. We might still be interested in natural questions about these packets, but we want to do this without storing all the packets.

24.5. Heavy hitters

The problem. Imagine a stream $s_1, \ldots,$ where elements might repeat, and we would like to maintain a list of elements that appear at least εn times, where $\varepsilon \in (0,1)$ is some parameter. The purpose here is to do this using as little space as possible.

24.5.1. A randomized algorithm

Am easy randomized algorithm, would maintain a random sample of size $m = \lceil (1/\varepsilon) \ln(1/\varphi) \rceil$, using reservoir sampling. The probability the sample fails to contain a heavy hitter after t insertions is

$$(1-\varepsilon)^m \leq \exp(-\varepsilon m) \leq \exp\left(-\varepsilon \left\lceil \frac{1}{\varepsilon} \ln \frac{1}{\varphi} \right\rceil\right) \leq \exp\left(-\ln \frac{1}{\varphi}\right) = \exp(\varphi) = \varphi.$$

24.5.2. A deterministic algorithm

Disclaimer: The following is a deterministic algorithm, but it is too elegant to hold this against it, and we will present it anyway.

The algorithm. To this end, let

$$k = \lceil 1/\varepsilon \rceil.$$

At each point in time, we maintain a set S of k elements, with a counter for each element. Let S_t be the version of S after t were inserted. When s_{t+1} arrives, we increase its counter if it is already in S_t. If

$|S_t| < k$, then we just insert s_{t+1} to the set, and set its counter to 1. Otherwise, $|S_t| = k$ and $s_{t+1} \notin S_t$. We then decrease all the k counters of elements in S_t by 1. If a counter of an element in S_{t+1} is zero, then we delete it from the set.

Correctness.

Lemma 24.5.1. *The above algorithm, after the insertion of t elements, the set S_{t+1} would contain all the elements in the stream that appears at least εt times.*

Proof: Conceptually, imagine that the algorithm keeps counters for all the distinct elements seen in the stream. Whenever a decrease of the counters happens – the algorithm decrease not k counters – but $k+1$ counters – the additional counter being a counter of the new element, which has value one, and goes down to zero. Clearly, the number of distinct remaining elements in any point in time is at most k – that is, the number of counters that have a non-zero value. Consider an element e that appears $u \geq \varepsilon t$ times in the stream. The counter for e is going to be increased u times, and decreased at most α time, where $\alpha(k+1) \leq t$. We have that the counter for u in the end of the stream must have value at least

$$u - \frac{t}{k+1} \geq \varepsilon t - \frac{t}{k+1} = \varepsilon t - \frac{t}{\lceil 1/\varepsilon \rceil + 1} \geq t \frac{\lceil 1/\varepsilon \rceil \varepsilon + \varepsilon - 1}{\lceil 1/\varepsilon \rceil + 1} \geq t \frac{\varepsilon}{\lceil 1/\varepsilon \rceil + 1} > 0.$$

This implies that the counter of u is strictly larger than 0, which implies that u appears in S_{t+1}. ■

24.6. From previous lectures

Lemma 24.6.1. *Let $X_1, \ldots, X_n$ be n independent Bernoulli trials, where $\mathbb{P}[X_i = 1] = p_i$, and $\mathbb{P}[X_i = 0] = 1 - p_i$, for $i = 1, \ldots, n$. Let $X = \sum_{i=1}^{b} X_i$, and $\mu = \mathbb{E}\left[X\right] = \sum_i p_i$. For $\delta, \in (0,1)$, we have*

$$\mathbb{P}\left[X > (1+\delta)\mu\right] < \exp\left(-\mu\delta^2/3\right).$$

References

[MP80] J. I. Munro and M. Paterson. *Selection and sorting with limited storage*. *Theo. Comp. Sci.*, 12: 315–323, 1980.

[MR95] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge, UK: Cambridge University Press, 1995.