M.I.T. — 6.045: Automata, Computability, and Complexity Handout
Ryan Williams Updated 2/26/2020

# Notes on Streaming Algorithms[1]

A *streaming* algorithm is an algorithm that receives its input as a "stream" of data, and that proceeds by making only one pass (or a small number of passes) through the data. As for any other kind of algorithm, we want to design streaming algorithms that are fast and that use as little memory as possible. A DFA can be viewed as a streaming algorithm that uses only a constant ($O(1)$) amount of memory, processes each data item (alphabet symbol) in its input stream in constant ($O(1)$) time, and solves a decision problem defined on the input. In general, we are interested in streaming algorithms for computations on numbers, and computations with other non-binary output, and we are interested in algorithms with memory usage and processing-time-per-input-item are not constants, as long as we can make them feasibly small even for very large streams. (For simplicity, here we will focus almost totally on the "low memory" aspect, and not talk much about the running time of our algorithms. Running time will come later in the class!)

The streaming model is appropriate for settings in which the data to be processed is not stored anywhere but it is generated dynamically, and is fed to the streaming algorithm as it is being generated. Typical examples are the stream of measurements from a sensor network, the stream of transactions going to and from an online bank, the stream of search strings in a search engine, of page requests coming to a web server, video and audio streams, and so on.

In these notes, we will see examples of streaming algorithms and their space usage. We will also see how to apply ideas from the Myhill-Nerode theorem and its proof to prove space lower bounds for streaming algorithms, showing both logarithmic lower bounds and linear lower bounds in certain cases.

## 1 The Streaming Model

More formally, we think of a streaming algorithm working over an alphabet $\Sigma$ as having three basic components.

- **Initialization**. This component initializes all the variables that will be used in the computation.
- **Update rule**. This component says, for every new symbol $\sigma \in \Sigma$ received from the stream, how the variables should change as a result.

[1]These notes began from the lecture notes of Luca Trevisan, while we were teaching CS154 at Stanford. Many of the sentences are his (probably most of the good ones).

- **Stopping rule**. This component tells us what to do when the stream ends: what is the accept/reject condition, or in general how the output should be derived.

In what follows, we won't really care about the efficiency / running time of these rules. What we will care about is *space usage*: the number of bits of memory that the streaming algorithm needs to compute the answer. In a situation where the data is of some length $n$, where $n$ is truly gigantic (e.g., $10^{19}$ or more), we would like the space usage of our streaming algorithm to be *significantly* less than $n$.

**Definition 1** *The* **space usage** *of a streaming algorithm $A$ is a function $S : \mathbb{N} \to \mathbb{N}$, where $S(n)$ is the maximum number of bits used to store the variables in $A$, over all inputs of length up to $n$.*

We say that a **streaming algorithm $A$ uses at most $S(n)$ space** if for all $n$, $A$ has space usage *at most* $S(n)$ on all inputs of length up to $n$.

We say that a **streaming algorithm $A$ uses at least $S(n)$ space** if for infinitely many $n$, $A$ has space usage *at least* $S(n)$ on all inputs of length up to $n$.

Given our extremely generic notion of streaming algorithm and space usage, essentially every interesting problem can be solved using at most $n$ space, by just storing the entire input in the algorithm. A space usage like $O(\sqrt{n})$ would be nicer, but something like $O(\log n)$ would be even better. (Note that a streaming algorithm with a space usage of $O(1)$ is equivalent to having a DFA!)

We will be particularly interested in cases where we can prove that there is a streaming algorithm with at most $a \cdot S(n) + a$ space for solving a problem $L$, and we can prove that every streaming algorithm uses at least $b \cdot S(n) + b$ space, for some constants $a \geq b > 0$. This means that we have found a streaming algorithm that is very close to the best possible for space usage; and if $a = b$, our algorithm really is the best possible! In such a case, we would say that $L$ has an $O(s(n))$-space streaming algorithm, and $L$ has an $\Omega(s(n))$-space streaming lower bound.

## 2 An Example

We start with a very simple streaming problem: *Given a string $x_1 \cdots x_n \in \{0,1\}^\star$, does it have more ones than zeroes?* We can formalize this by considering the language

$$\textsc{Majority} = \{x \mid x \text{ has more 1s than 0s}\}.$$

There is a simple streaming algorithm for Majority with modest space usage:

- **Initialize:** $C := 0$ and $B := 0$
- **When next symbol seen is $\sigma$:**
  If $(C = 0)$ then $B := \sigma$, $C := 1$
  If $(C \neq 0)$ and $(B = \sigma)$ then $C := C + 1$
  If $(C \neq 0)$ and $(B \neq \sigma)$ then $C := C - 1$
- **When stream stops:** if $B = 1$ and $C > 0$ then *accept*, else *reject*.

(For an intuitive explanation about why this works, see the slides.)

What is the space usage of this algorithm? Suppose the integer in $C$ is stored in binary. We need one bit to store $B$, and we need at most $\log_2(n) + O(1)$ bits to store the binary counter.

## 2.1 Proving a lower bound

Maintaining a simple counter is easy; could we somehow use less memory, and still solve the MAJORITY problem? And if not, how could we *prove* that? Here we will show how to import ideas from the Myhill-Nerode theorem, and prove:

**Theorem 2** *For all even $n$, every streaming algorithm computing* MAJORITY *needs to use at least* $\log_2(n)$ *bits of space.*

The argument can be modified to work for odd $n$ as well, it is just a little more technical. Let's introduce the lower bound framework for streaming algorithms.

**Proving lower bounds on streaming.** We first introduce an analogue of distinguishing sets, as seen in the Myhill-Nerode theorem.

**Definition 3** *Let $L \subseteq \Sigma^\star$ and let $n \in \mathbb{N}$. A* **streaming distinguisher for $L_n$** *is a subset $S_n$ of $\Sigma^\star$ where, for every distinct $x, y \in S_n$, there is a $z \in \Sigma^\star$ such that $|xz| \leq n$, $|yz| \leq n$, and exactly one of $xy$, $yz$ is in $L$.*

Observe that the only difference between a "streaming distinguisher" and a "distinguishing set" from the Myhill-Nerode theorem, is the requirement that the strings $xz$ and $yz$ all be of length at most $n$. This is important for us, because we care about how the space usage (memory states) of the streaming algorithm grows, as the lengths of inputs grow. Note that this extra length requirement means that $S_n$ contains only strings of length at most $n$.

Now we are ready to state our main theorem for streaming lower bounds:

**Theorem 4 (Streaming Lower Bound Theorem)** *Suppose for all $n$, there is a streaming distinguisher $S_n$ for $L_n$ with $|S_n| \geq 2^{S(n)}$. Then every streaming algorithm for $L$ uses at least $S(n)$ space on inputs of length $\leq n$.*

Proof: For every $n$, let $S_n = \{x_1, \ldots, x_k\}$ be a streaming distinguisher for $L_n$, with $|S_n| \geq 2^{S(n)+1}$. Let $A$ be a streaming algorithm for $L$, and let $m_i$ be the memory state that $A$ is in, after reading in $x_i$.

We claim that $m_i \neq m_j$ for all $i \neq j$. Suppose for contradiction that $A$ reaches the same memory state $m$ after reading in both $x_i$ and $x_j$. Then for any string $z$, $A$ must give the same output on both $x_i y$ and $x_j z$. However, by definition of $S_n$, there is a string $z$ such that *exactly one* of $x_i z$, $x_j z$ is in $L$. Therefore $A$ must give an incorrect answer on at least one of $x_i z$, $x_j z$, a contradiction.

Since all of the memory states $\{m_1, \ldots, m_k\}$ are distinct, and all strings $x_i z$ and $x_j z$ read among those memory states have length at most $n$, the algorithm $A$ must reach at least $|S_n| \geq 2^{S(n)}$ distinct memory states, over the strings of length up to $n$. Now, if $A$ used at most $S(n) - 1$ space on inputs of length $\leq n$, then $A$ would only have at most $\sum_{i=0}^{S(n)-1} 2^i = 2^{S(n)} - 1$ distinct memory states over those inputs. Therefore $A$ must use at least $S(n)$ space. $\square$

Rather than proving a lower bound for Majority here (unfortunately, we slipped up and put something like that on the pest this week), I will prove a related lower bound for another problem that we know to not be regular. (I'll probably add the Majority lower bound later on.)

**Theorem 5** *For all even $n$, every streaming algorithm computing $L = \{0^k 1^k \mid k \geq 0\}$ must use at least $\lfloor \log_2(n) \rfloor$ bits of space.*

Proof: For all even $n$, we give a streaming distinguisher $S_n$ for $L_n$ with $|S_n| \geq n/2 + 1$. By Theorem 4, this implies that for all even $n$, every streaming algorithm for $L$ uses at least $\log_2(n/2 + 1)$ space. Note that

$$\log_2(n/2 + 1) > \log_2(n/2) = \log_2(n) - 1,$$

and space usage must be an integer, so the space usage is at least $\lfloor \log_2(n) \rfloor$.

To get the streaming distinguisher, we simply define

$$S_n = \{0^i \mid i = 0, 1, \ldots, n/2\}.$$

Consider any two strings $0^a$ and $0^b$ from $S_n$. Set $z = 1^b$. Then $0^a 1^b \notin L$, $0^b 1^b \in L$, and both strings are of length at most $n$. Therefore $S_n$ is a streaming distinguisher for $L_n$, and the proof is complete. $\square$

**Remark 6** *We can improve the above space lower bound, by choosing the slightly larger set $S_n = \{0^i 1^j \mid i = 0, 1, \dots, n/2, j = 0, 1\}$. Then $|S_n| = 2(n/2 + 1) = n + 2$. Given two strings $0^i$ and $0^{i'}$ with $i \neq i'$, we can distinguish them as in the previous proof. Given any $x = 0^i$ and $y = 0^{i'}1$, even if $i = i'$, we can distinguish $x$ and $y$ with the string $01^{i+1}$, because $x01^{i+1} = 0^{i+1}1^{i+1}$ is in $L$, but $y01^{i+1} = 0^{i'}101^{i+1}$ is not in $L$. Therefore the above space bound can be further improved to $\log_2(n + 2)$.*

Note it is easy to get a streaming algorithm with $\log_2(n) + O(1)$ space usage for the above $L$, by simply keeping a binary counter of the 0's, and compare that count to the number of 1's. (The slides give precise pseudocode.)

## 3 Most Frequent Element

Another example of a simple but non-trivial streaming algorithm is given by the following typical interview question: suppose we are allowed to make one pass through a sequence $x_1, \dots, x_n$ of elements from a set $\Sigma$, and that we know that one value from $\Sigma$ occurs more than $n/2$ times in the sequence; find the most frequently repeated value using only two variables, using a total memory of only $\log_2 n + \log_2 |\Sigma|$ bits. (If you haven't seen this puzzle before, think about how you would do it.)

The problem of finding a most frequently occurring element (MFE) in a data stream is an important one in many of the settings that motivate the streaming model. (Think of keeping track of the page that receives the most hits, or the best selling item, etc.) In general, that is, without the guarantee that there is an element occurring in a majority of places in the sequence, there is, unfortunately, no memory-efficient way to find a most frequent element.

We will show that every deterministic streaming algorithm that solves the MFE problem must use at least $\Omega(n \cdot \ell)$ bits of memory, where $n$ is the length of the strings and $\ell = \log_2 |\Sigma|$ is the bit-length of one element of $\Sigma$. We will assume $2^\ell > n^2$.

We now give a generic definition of distinguishability for streaming problems.

**Definition 7** *We say that two streams $\mathbf{x}$, $\mathbf{y}$ are distinguishable for a streaming problem $P$ on inputs of length $n$, if there is a stream $\mathbf{z}$ such that all the correct answers to problem $P$ for input $\mathbf{x} \cdot \mathbf{z}$ are different from all the correct answers to problem $P$ for input $\mathbf{y} \cdot \mathbf{z}$, and the streams $\mathbf{x} \cdot \mathbf{z}$ and $\mathbf{y} \cdot \mathbf{z}$ have length $n$.*

For example, two streams $\mathbf{x}$, $\mathbf{y}$ are distinguishable for MFE if there is a stream $\mathbf{z}$ such that no "most frequent element" of $\mathbf{x} \cdot \mathbf{z}$ is also a "most frequent element" of $\mathbf{y} \cdot \mathbf{z}$.

The following fact gives us a generic way to prove memory lower bounds.

**Lemma 8** *Suppose that we can find $D(n,\Sigma)$ strings in $\Sigma^*$ that are distinguishable for problem $P$ on inputs of length $n$. Then, every deterministic streaming algorithm for $P$ must use $\geq \log_2 D(n,\Sigma)$ bits of space on inputs of length $n$.*

Proof: Suppose there is a streaming algorithm $A$ using $m(n,\Sigma) < \log_2 D(n,\Sigma)$ bits of memory that solves $P$ on inputs of length $n$. Then $A$ has $\leq 2^{m(n,\Sigma)} < D(n,\Sigma)$ distinct internal states, and there are two distinguishable strings $\mathbf{x}$ and $\mathbf{y}$ such that $A$ is in the same internal state after reading $\mathbf{x}$ and after reading $\mathbf{y}$. This means that, for every $\mathbf{z}$, $A$ gives the same output for the input $\mathbf{x}\cdot\mathbf{z}$ and the input $\mathbf{y}\cdot\mathbf{z}$. So, for every $\mathbf{z}$, there must be an output that is correct both for the input $\mathbf{x}\cdot\mathbf{z}$ and the input $\mathbf{y}\cdot\mathbf{z}$, contradicting the distinguishability of $\mathbf{x}$ and $\mathbf{y}$. $\Box$

Now we turn to applying Lemma 8 to the MFE problem.

**Definition 9** *For every $n$, $\Sigma$, define the language $L_{n,\Sigma} \subseteq \Sigma^n$ of sequences $x_1,\ldots,x_n$, such that $x_i \in \Sigma$ for all $i$, and $0$ is a most frequently occurring element.*

**Lemma 10** *There are $2^{\Omega(n\ell)}$ distinguishable strings for MFE on inputs of length $n$.*

Proof: For every subset $S = \{s_1,\ldots,s_{(n-5)/2}\} \subset \Sigma - \{0\}$ of $\frac{n-5}{2}$ nonzero elements of $\Sigma$, consider the sequence

$$\mathbf{x}_S := 0,0,0,s_1,s_1,\ldots,s_{\frac{n-5}{2}},s_{\frac{n-5}{2}}$$

of length $n-2$ where $0$ is repeated 3 times, the elements of $S$ are repeated twice each, and no other element is present.

All such sequences are *distinguishable* strings, because if we consider any two different sequences $\mathbf{x}_S$ and $\mathbf{x}_T$, and we take an element $s$ which belongs to $S$ but not to $T$, we see that attaching $(s,s)$ to $\mathbf{x}_S$ gives us a sequence not in $L_{n,\Sigma}$ (because the most frequent element is $s \neq 0$, which occurs 4 times), while attaching $(s,s)$ to $\mathbf{x}_T$ gives us an element of $L_{n,\Sigma}$, because 0 occurs 3 times and all other elements occur only once.

The number of distinguishable strings that we have constructed is

$$\binom{2^\ell - 1}{\frac{n-5}{2}} \geq \left(\frac{2^\ell - 1}{e\cdot\left(\frac{n-5}{2}\right)}\right)^{\frac{n-5}{2}} \geq 2^{\Omega(n\ell)}$$

where we use the fact that $\binom{N}{K} \geq \left(\frac{N}{eK}\right)^K$ and the assumption that $2^\ell > n^2$, so that

$$\left(\frac{2^\ell - 1}{e\cdot\left(\frac{n-5}{2}\right)}\right) \geq \Omega(2^{\ell/2})$$

This completes the proof. $\Box$

Putting together Lemma 8 and Lemma 10, we have shown:

**Theorem 11** *Every deterministic streaming algorithm for the MFE problem requires memory $\Omega(n\ell)$, where $n$ is the length of the stream and $\ell$ is the bit-length of each data item, assuming $2^\ell > n^2$.*

## 4 Number of Distinct Elements

Another useful computation to make over a data stream is to figure out how many *distinct elements* (DE) there are in the stream. For example, from a stream of page requests we would like to know from how many distinct IP address we are getting requests, as a first approximation of the number of unique readers.

**For simplicity, we will continue to work with the assumption $|\Sigma| = 2^\ell > n^2$.**

To prove lower bounds for this problem, we will reason directly about the streaming algorithm as before. We say that two streams $\mathbf{x}$, $\mathbf{y}$ are distinguishable for the DE problem if there is a stream $\mathbf{z}$ such that the number of distinct elements in $\mathbf{x} \cdot \mathbf{z}$ is different from the number of distinct elements of $\mathbf{y} \cdot \mathbf{z}$.

**Theorem 12** *There are $2^{\Omega(n\ell)}$ distinguishable strings for the DE problem on inputs of length $n$, hence the DE problem requires $\Omega(n\ell)$ bits of memory for every deterministic streaming algorithm.*

Proof: For every subset $S = \{s_1, \ldots, s_{n/2}\} \subseteq \Sigma$ of size $n/2$, consider the sequence $\mathbf{x}_S = s_1, \ldots, s_{n/2}$. For every two sequences $\mathbf{x}_S$ and $\mathbf{x}_T$, we can see that they are distinguishable, because $\mathbf{x}_S \cdot \mathbf{x}_S$ has $n/2$ distinct elements, and $\mathbf{x}_T \cdot \mathbf{x}_S$ has strictly more than $n/2$ distinct elements.

The number of distinguishable strings we have constructed is

$$\binom{2^\ell}{\frac{n}{2}} \geq \left(\frac{2 \cdot 2^\ell}{n}\right)^{n/2} \geq 2^{\Omega(n\ell)},$$

where we have used the estimate $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$ and the fact that $2^k > n^2$. $\Box$

What about *approximating* the number of distinct elements?

**Theorem 13** *There are $2^{\Omega(n\ell)}$ distinguishable strings for the problem of approximating DE within a $\pm 20\%$ relative error on inputs of length $n$, hence the DE problem requires $\Omega(n\ell)$ bits of memory for every deterministic streaming algorithm.*

Proof: We will use the following fact without proof: under the usual assumption $2^\ell > n^2$, there is a collection $\mathcal{S}$ of $2^{\Omega(nl)}$ subsets of $\Sigma$, each of size $n/2$, and such that every two sets $S, T \in \mathcal{S}$ have at most $n/10$ elements in common.

Now, for every set $S \in \mathcal{S}$, consider the sequence $\mathbf{x}_S = s_1, \ldots, s_n/2$. For every two sequences $\mathbf{x}_S$ and $\mathbf{x}_T$, we can see that they are distinguishable, because $\mathbf{x}_S \cdot \mathbf{x}_S$ has $n/2$ distinct elements, and so the range of possible answers of a 20%-approximate algorithm is between $.4n$ and $.6n$, while $\mathbf{x}_T \cdot \mathbf{x}_S$ has at least $.9n$ distinct elements and so the range of valid answers is between $.72n$ and $1.08n$, so no answer can be valid for both sequences. $\square$

## 5 Randomness Helps! (Optional Material)

The DE problem, however, admits very efficient *randomized* approximate streaming algorithms. For example, it is possible to achieve a 1% approximation with high probability using only $O(\ell + \log n)$ bits of memory.

The basic idea is to randomly pick a hash function $h : \Sigma \to [0, 1]$ that randomly maps data items to real numbers in the range $[0, 1]$. Then, given a sequence $x_1, \ldots, x_n$, we compute $h(x_i)$ for each $i$, and store the *minimum* hash value $m$; the output is $1/m$.

The point of the algorithm is that (assuming $h$ to be a perfectly random hash function), the process of evaluating $h(x_i)$ for each $i$ and defining $m$ to be the minimum is the same probabilistic process as picking $k$ random real numbers in $[0, 1]$, where $k$ is the number of distinct elements in $x_1, \ldots, x_n$, then defining $m$ to be the minimum.

The latter process is well understood, and $m$ tends to be approximately $1/k$, so that $1/m$ is an approximation to $k$.

Here is a simplified version of the analysis: we want to show that there is a good probability that the minimum of $k$ random real numbers in the range $[0, 1]$ is $\Omega(1/k)$ and $O(1/k)$. First, we see that the probability that the minimum is more than $5/k$ is

$$\left(1 - \frac{5}{k}\right)^k \leq e^{-5} < 0.007$$

and the probability that the minimum is less than $1/(10k)$ is at most $1/10$.

The storage required to implement the algorithm is the memory used to store $h$, plus the memory needed to store the current minimum. Real numbers are represented with finite precision, which affects the algorithm negligibly, and $h$ is picked not as a completely random function (which would require storage space proportional to $|\Sigma|$) but as “pairwise independent” hash function, which requires storage $O(\log |\Sigma|)$. The analysis of the completely random case needs to be adjust to deal with the more limited randomness property of the hash function used in the implementation.

Both the probability of finding a good approximation and the range of approximation can be improved with various techniques.