Notes on Streaming Algorithms

A *streaming* algorithm is an algorithm that receives its input as a "stream" of data, and that proceeds by making only one pass 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 is a streaming algorithm that uses a constant amount of memory, processes each data item in its input stream in constant time, and solves a decision problem defined on the input. In general, we are interested in streaming algorithms for numerical computations, and other computations with a non-binary output, and we are interested in algorithm whose memory use and processing-time-per-input-item are not constants, as long as we can make them feasibly small even for very large streams.

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

In this lecture we will see how to apply the Myhill-Nerode theorem and the ideas in its proof to argue about memory lower bounds for streaming algorithms.

1 Most Frequent Element

An 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 $x_1, \ldots, x_n$ of elements from a set Σ, and that we know that one value from Σ 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 Σ. We will assume $2^\ell > n^2$.

We will prove the memory lower bound by relating the amount of memory needed by a streaming algorithm for MFE to the number of states needed by any DFA for a related problem. Let's identify Σ with the set $\{0, \ldots, 2^\ell - 1\}$.

Definition 1 *For every n, Σ, define the language $L_{n,\Sigma} \subseteq \Sigma^n$ of sequences $x_1, \ldots, x_n$, $x_i \in \Sigma$ in which 0 is a most frequently occurring algorithms.*

Fact 2 *Suppose that there is a deterministic streaming algorithm for MFE that uses $\leq m(n, \Sigma)$ bits of memory; then there is a DFA for $L_{n,\Sigma}$ that has $\leq 2^{m(n,\Sigma)}$ states.*

PROOF: A streaming algorithm that uses m bits of memory has only 2^m distinct internal states, and so we can construct an automaton with 2^m states that can simulate the execution of the algorithm; at the end of sequence, we define the internal states that lead the algorithm to output 0 are accepting states for the DFA, the others are not accepting. This automaton recognizes the language $L_{n,\Sigma}$. $\square$

Now we can apply the Myhill-Nerode theorem to $L_{n,\Sigma}$.

Lemma 3 *The language $L_{n,\Sigma}$ has $2^{\Omega(n\ell)}$ distinguishable strings.*

PROOF: For every subset $S = \{s_1, \ldots, s_{(n-5)/2}\} \subset \Sigma - \{0\}$ of $\frac{n-5}{2}$ nonzero elements of Σ, 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.

We see that all such sequences are *distinguishable* strings for $L_{n,\Sigma}$, 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

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

□

Putting all together we have.

Theorem 4 *Every deterministic streaming algorithm for the MFE problem requires memory $\Omega(n\ell)$, where n is the length of the stream and ℓ is the bit-length of each data item, assuming $2^\ell > n^2$.*

2 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, instead of reducing to a regular language and finding distinguishable strings for it, we will reason directly about the streaming algorithm.

Definition 5 *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.*

So two streams $\mathbf{x}$, $\mathbf{y}$ are distinguishable for the MFE problem if there is a stream $\mathbf{z}$ such that none of the most frequent elements of $\mathbf{x} \cdot \mathbf{z}$ is also a most frequent element of $\mathbf{y} \cdot \mathbf{z}$. 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 element of $\mathbf{y} \cdot \mathbf{z}$.

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

Lemma 6 *Suppose that we can find $D(n, \Sigma)$ strings in Σ^* that are distinguishable for problem P on inputs of length n. Then, every deterministic streaming algorithm for P must use $\geq \log_2 D(n, \Sigma)$ memory on inputs of length n.*

Proof: Suppose that we have a streaming algorithm A that uses $m(n, \Sigma) < \log_2 D(n, \Sigma)$ bits of memory and 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 $\mathbf{y}$. □

Note that we could have derived the lower bound for MFE from Lemmas 3 and 6 without reducing to a regular language.

Theorem 7 *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

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

□

What about *approximating* the number of distinct elements?

Theorem 8 *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 Σ, 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. □

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 reals in the range of 0 to 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.