CS Theory (Spring '25) Fall 2025

Lecture Notes: DFAs and Streaming Algorithms

Instructor: *Toniann Pitassi*

Until now, we introduced several computation models, including DFA, NFA, and regular expressions, but all of them only recognize regular languages. Today we introduce streaming algorithms, which is an important model of computation that is able to process large quantities of data efficiently, and crucially, using very little memory. [1] We will see that constant-space streaming algorithms are another model of computation that is *equivalent* to DFAs! That is, we will show that constant-space streaming algorithms can recognize exactly the same class of languages as regular languages. We first motivate streaming algorithms informally and then proceed with the formal definition.

For streaming algorithms, each step the algorithm can read one input symbol. In general the input alphabet can be any finite set Σ, just as it was for DFAs, but we will primarily focus on $\Sigma = \{0, 1\}$. In this case, each step of a streaming algorithm reads one bit of the input, and after reading a bit, the algorithm cannot backtrack and see it again. Obviously, we can always store all the data we read, and then do what we want on the data. However, a typical scenario is that the input size is so big that we do not want to store the whole inputs, but we still want to figure out interesting things about the data. We want to use as little space as possible (while ensuring that we still have the correct results), and we will mainly measure the performance of the streaming algorithms based on its space usage.

We first give some real-life examples as motivation. Large search engines such as Google receive roughly 10 million searches per minute! Google wants to process this enormous stream of information to obtain statistics and trends in a timely manner. This requires efficient computation in terms of both time and storage. With this much data it is simply not feasible to store all of it in raw form, and secondly it is desirable to not store it as a first step towards maintaining individual's privacy.

Another example is the training of large language models (LLMs), which are trained on an enormous corpa of data. To get a sense of just how large this is, ChatGPT3 (2020) was trained on roughly 570 gigabytes (GB) of text, and each new generation has used roughly 2 times the amount of training data as the previous generation. According to ChatGPT5: "while no official figure exists, GPT-5 almost certainly uses tens of trillions of tokens (vs GPT-3's hundreds of billions), which works out to many petabytes of raw data." Some other examples include processing of experimental data from measurement devices: the device may generate lots of data in every second, but we only want to extract certain useful information.

1 Streaming Algorithms

Now we give the formal definition of streaming algorithms.

Definition 1 (Streaming Algorithms)**.** *A streaming algorithm $\mathcal{A} = (\Sigma, \Pi, I, \delta, \gamma)$ is described by a 5-tuple. The algorithm has an underlying input alphabet Σ and an underlying memory alphabet*

[1] T. Pitassi gratefully acknowledges Josh Alman for sharing his lecture notes from CSTheory, Spring 2025. This sequence of 4 lecture notes are a revised version of Alman's notes.

Π, *where* $\Sigma \subseteq \Pi$. *Both* Σ *and* Π *are finite alphabet symbols, and we will usually assume that they are both* $\{0, 1\}$. *A streaming algorithm has a working memory* $M \in \Pi^*$ *which can be viewed as the state of the computation. At any point in time, the memory contains different kinds of information about the input seen so far. During the execution of the algorithm, the contents of memory changes to reflect what information the algorithm is keeping track of. The input to a streaming algorithm is a string* $w \in \Sigma^*$ *(just like the iinput to a DFA). There are three components to describe a streaming algorithm:*

- *An initialization rule* $\mathcal{I} \in \Pi^*$. *This tells the algorithm the value of* M *at the start of the computation, before any bits of the input are read. Typically* $\mathcal{I} = \epsilon$; *that is, at the start* M *consists of the empty string.*
- *An update rule* $\delta : \Pi^* \times \Sigma \to \Pi^*$. *This tells the algorithm how to update the memory after reading the next symbol of the input string.*
- *A stopping rule* $\gamma : \Pi^* \to \mathrm{OUT}$. *This tells the algorithm what to output at the end of the execution (after reading all data), depending on the content of the memory at that time. Here* OUT *is the set of possible outputs of the algorithm, and this would depend on the task we are trying to solve. In this class, we will restrict attention to streaming algorithms for decision problems and thus* OUT *will be 0 (reject) or 1 (accept).*

Finally, a streaming algorithm $\mathcal{A}$ *accepts a language* $L \subseteq \{0, 1\}^*$ *if and only if for every input* $w \in \{0, 1\}^*$ *if* $w \in L$ *then* $\mathcal{A}$ *outputs* 1 *(accept) on input* w, *and if* $w \notin L$, *then* $\mathcal{A}$ *outputs* 0 *(reject). Just like DFAs and NFAs, for every streaming algorithm (where the output is either 0 or 1), there is a unique language associated with it.*

We focus on space usage of the streaming algorithms which we define next.

Definition 2 (Space usage)**.** *The space usage of a streaming algorithm* A *is a function* $S : \mathbb{N}_{\geq 0} \to \mathbb{N}_{\geq 0}$ *where* $S(n)$ *is the maximum number of bits used to store* M *(that is the maximum length of* M*), over all possible inputs of length at most* n.[2]

In general we will often be representing integers by their binary representation. In the examples below, we will use the following notation to go back and forth between a number and its binary representation.

Definition 3 (Integers and their binary representation)**.** *For a nonnegative integer* x, *let* $\langle x \rangle$ *denote the binary representation of* x. *Conversely, for a string* $w \in \{0, 1\}^*$, *let* $num(w)$ *denote the nonnegative integer represented by* w. *For an integer* x *(that can be positive or negative), we will represent* x *by a string* $\langle x \rangle \in \{-, 0, 1\}^*$ *of the following form: If* $x \geq 0$, *then* $\langle x \rangle$ *is just the binary representaton of* x; *if* $x < 0$, *then* $\langle x \rangle = -u$ *where* u *is the binary representation of* x.

For example, if $x = 5$, $\langle x \rangle = 101$, *and if* $x = -7$, $\langle x \rangle = -111$. *Conversely,* $num(101) = 5$, *and* $num(-111) = -7$.

We will now see a couple of examples.

Example 4. *Let* $\mathrm{MAJ} = \{w \in \{0, 1\}^* \mid w$ *contains at least as many 1's as 0's*$\}$. *A streaming algorithm that computes it is as follows: The input alphabet is* $\Sigma = \{0, 1\}$ *and the memory alphabet is* $\Pi = \{0, 1, 2\}$. *At every point in time the contents of memory,* $M \in \{0, 1, 2\}^*$, *will contain a string of the form* $u2v$ *where* u *is the binary representation of the number of 0's seen so far, and* v *is the binary representation of the number of 1's seen so far. The symbol 2 is put in the middle so that we can decode the string to get back* u *and* v.

[2] $\mathbb{N}_{\geq 0}$ is the set of non-negative integers.

- *Initialization:* $M = 020$ *since initially we have not seen any 0's or any 1's, so* M *has the form* $u2v$ *where both* u *and* v *are the binary representations of 0.*

- *Update rule* δ*: Let* $M = u2v$*. If* $\sigma = 0$*, then* $\delta(M, \sigma) = u'2v$ *where* $u' = \langle num(u) + 1\rangle$*. Otherwise if* $\sigma = 1$*, then* $\delta(M, \sigma) = u2v'$ *where* $v' = \langle num(v) + 1 >$*.*

- *Stopping rule: Let* $M = u2v$*. If* $num(u) \leq num(v)$*, then let* $\gamma(M) = 1$ *(output 1); otherwise* $\gamma(M) = 0$ *(output 0).*

We now consider the space usage of the above algorithm. We will consider the usage in terms of $n = |w|$, which is the number of input bits, since we are concerned with how the space usage grows when the input size grows. For an input string $w \in \{0,1\}^*$, the number of 1's in w is at most n and similarly the number of 0's in w is at most n. Since we express these two numbers in binary, u and v will have length at most $\lceil \log_2 n\rceil$. Therefore the length of $M = u2v$ on any input w, $|w| = n$, is at most $\lceil 2\log_2 n\rceil + 1 = O(\log n)$.

Example 5. *Let* $L = \{w \in \{0,1\}^* \mid$ *number of* 1*'s in* w *is divisible by* $4\}$*. A streaming algorithm that computes it is as follows: The input alphabet is* $\Sigma = \{0,1\}$ *and the memory alphabet is* $\Pi = \{0,1\}$*.*

- *Initially,* $M = 0$ *(this is the binary representation of 0).*

- *Update rule: on input* $\sigma \in \{0,1\}$*, If* $\sigma = 0$ *then* M *isn't updated. Otherwise if* $\sigma = 1$*, update* M *to be* $\langle num(M) + 1 \pmod 4\rangle$*, the binary representation of* $M + 1 (\mathrm{mod} 4)$*. Examples when* $\sigma = 1$*: if* $M = 0$*, them update* $M = 01$*; if* $M = 10$ *then update* $M = 11$*; if* $M = 11$ *then update* $M = 0$*.*

- *Stopping rule: if* $M = 0$*, output 1; else output 0.*

In the algorithm above, M always keep track of the number of 1's in the portion it has already read modulo 4. Since the possible values of M are $0, 01, 10, 11$, M needs 2 bits for any length input, so the space usage of the algorithm is constant.[3]

Definition 6. *A streaming algorithm* $\mathcal{A}$ *uses constant space if* $S(n) = O(1)$*. I.e. there exists a constant* c *such that for all* $w \in \Sigma^*$*, the memory* M *always has length* $|M| \leq c$*.*

It is not hard to see that L is a regular language. Below we will prove that regular languages are exactly those languages that can be computed by streaming algorithm with constant space usage.

Theorem 7. *Let* L *be a regular language and let* M *be a DFA that recognizes* L*. If* M *has* S *states, then there is a streaming algorithm that recognizes* L *with space* $\lceil \log S\rceil = O(\log S)$*.*

Proof. Since the language is regular, it is recognized by a DFA, M. Suppose that M has S states: $\mathcal{D} = (Q = \{q_0, \ldots, q_{S-1}\}, \Sigma, \delta, q_0, F)$. We construct the following streaming algorithm $\mathcal{A}$ that recognizes the same language as M as follows: The input alphabet is $\Sigma = \{0,1\}$ and the memory alphabet is $\Pi = \{0,1\}$. The main idea is that the streaming algorithm will simulate the DFA M, where the contents of memory will store the name of the state that M is in at each point in time. So if M is in state q_i at some point in the computation, then the contents of memory at the same point in time in $\mathcal{A}$ will be $M = \langle i\rangle$, the binary representation of i.

- Initialization: set $M = \langle 0\rangle$ (since the start state of M is q_0)

[3] We usually call a function *constant* if the function is $O(1)$ (thus bounded by a constant).

- Update rule δ': Let $\sigma \in \{0,1\}$ be the next current input symbol read, and suppose that $\delta(q_{num(M)}, \sigma) = q_i$. Then $\delta'(M, \sigma) = \langle i \rangle$; that is, the new M should be the name of the state that we are in after reading σ from the current state, $q_{num(M)}$.
- Stopping rule: if $q_{num(M)} \in F$, output 1; else output 0.

We can see that the algorithm $\mathcal{A}$ just simulates the DFA $\mathcal{D}$ on $\mathcal{A}$'s input: the current state q_i at each point in time in the execution of $\mathcal{D}$ corresponds to the contents of memory $M = \langle i \rangle$ at that same point in the simulation. If on an input w, the final state we are in when running $\mathcal{D}$ on w is an accept state, then the contents of the memory in the streaming algorithm at the end will also correspond to an accept state, so both the DFA and the streaming algorithm will accept. Conversely, if $\mathcal{D}$ ends up in a non-accept state, then the contents of memory at the end of the streaming algorithm will also be a non-accept state, so both will reject w. Therefore, $\mathcal{A}$ accepts if and only if the input is in the language. ■

Theorem 8. *If a language L is computed by a streaming algorithm with constant space s, then L is recognized by a DFA with $2^{s+1} - 1 = O(2^s)$ states.*

Proof. Denote by $\mathcal{A} = (\Sigma, \Pi = \{0,1\}, I, \delta, \gamma)$ the streaming algorithm. Note that since $\mathcal{A}$ uses space s, we always have that the memory M takes some value $m \in \Pi^*$ with $|m| \leq s$. The goal of our DFA will be to keep track of the memory M of the streaming algorithm as it reads symbols from w. As such we let $Q = \{q_m \mid m \in \Pi^* \text{ and } |m| \leq s\}$. We have that $|Q| = \sum_{i=0}^{s} 2^i = 2^{s+1} - 1$.

We define a DFA $\mathcal{D} = (Q, \Sigma, \delta', q_I, F)$ as follows.

- The set of states of D is Q.
- The transition function is just δ', where $\delta'(q_m, \sigma) = q_{m'}$ if $\delta(m, \sigma) = m'$. That is, if the streaming algorithm updates $M = m$ to $M = m'$ upon seeing σ, the the we update the state q_m to $q_{m'}$ when seeing σ.
- The start state is q_I, where I is the initial configuration of M.
- The set of accepting states $F = \{q_m \in Q \mid \gamma(m) = 1\}$. That is state q_m is an accept state, if when the streaming algorithm halts with $M = m$ it outputs 1 (Accept).

We can see (and formally prove by induction on the length of the input w) that if the algorithm $\mathcal{A}$ and the DFA $\mathcal{D}$ read the same input $w = w_1 w_2 \ldots w_t$, the current setting of the memory of $\mathcal{A}$ is $M = m$ if and only the current state of $\mathcal{D}$ is q_m. Suppose the final setting of the memory of $\mathcal{A}$ is m^*, then the final state of $\mathcal{D}$ is also q_{m^*}. Therefore, $\mathcal{A}$ accepts if and only if $\gamma(m^*) = 1$ (Accept), which is equivalent to $q_{m^*} \in F$. Thus, $\mathcal{D}$ recognizes the same language as $\mathcal{A}$, so $\mathcal{D}$ recognizes L and L is regular. ■

The above two theorems together imply the following corollary, stating that regular languages can equivalently be characterized by constant-space streaming algorithms:

Corollary 9. *A language $L \subseteq \{0,1\}^*$ is regular if and only if it is recognized by a streaming algorithm that uses space $O(1)$ (that is, the streaming algorithm uses constant space).*

To conclude, we have introduced the streaming model of computation, and proved that the class of languages that can be recognized by constant-space streaming algorithms are exactly the class of regular languages (that is, the class of all languages that can be recognized by some DFA). Thus we have now given four different definitions that are all equivalent to regular languages: (i) languages recognizable by a DFA; (ii) languages recognizable by an NFA; (iii) languages expressible by a regular expression; and (iv) languages recognizable by a constant-space streaming algorithm.

In the next class we will develop techniques for showing that not all languages are regular, and prove that some particular languages are not regular.