CS Theory (Spring '25) February 20, 2025

Lecture Note: Streaming Algorithms

Instructor: *Josh Alman*

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 can recognize more than just regular languages. In fact, we will prove that streaming algorithms can recognize all languages.

For streaming algorithms, in each step the algorithm can read a piece of input (usually a constant number of bits), and after reading a piece, the algorithm cannot 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.

Example 1. *On* `https://trends.google.com` *(Figure 1), we can see the recent trending searches. There are billions of searches on Google every day, and Google wants to process them to show the trending searches, while not storing all the searches.*

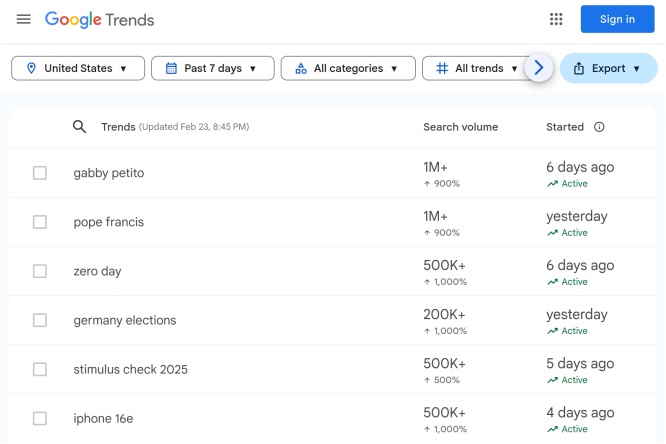

Figure 1: A screenshot of Google Trends

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

Example 2. *Let* $L_2 := \{w \in \{0,1\}^* \mid w$ *contains more* 0*'s than* 1*'s*$\}$*. A streaming algorithm that computes it is as follows:*

- *Variables: the algorithm maintains two variables a, b.*
- *Initialization: at the beginning it sets $a \coloneqq 0$, and sets $b \coloneqq 0$.*
- *Update rule: when the algorithm reads the next piece of data (in this case a symbol) $\sigma \in \{0, 1\}$, if $\sigma = 0$, then it lets $a \coloneqq a + 1$, otherwise it lets $b \coloneqq b + 1$.*
- *Stopping rule: after the algorithm reads the last piece of data, it accepts if $a > b$, and rejects otherwise.*

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.

The algorithm above stores only 2 variables, but its space usage is not constant. This is because the variables a and b may be as large as n. We can store a non-negative integer that is at most n using $\lceil \log_2(n+1) \rceil = O(\log n)$ bits by storing it in its binary form,[1] so the space usage of the above algorithm is $O(\log n)$.

Remark 3. To store a non-negative integer that is smaller than n, we need at least $\log_2 n$ bits of space. This is because there are 2^t possible configurations for t bits of storage, and each possible value of the integer must correspond to a distinct configuration. Therefore, n, the number of possible values, is at most 2^t.

Next time, we will prove we cannot do much better in terms of space usage. The intuition is as follows. Suppose w is first a lot of 0's and then about the same number of 1's. Intuitively, you need to keep track of the exact number of 0's to see whether there are more 0's or more 1's.

Now we give the formal definition of streaming algorithms.

Definition 4 (Streaming Algorithms)**.** A streaming algorithm $\mathcal{A}$ over the alphabet Σ has a working memory $M \in \Pi^*$ for some set Π. The memory can be viewed as a set of variables $M = \{X_1, \ldots, X_\ell\}$, and Π is the set of possible values of these variables. These variables can store different kinds of information, and their values might change during the execution of the algorithm. The algorithm has the following three components:

- An initialization rule $\mathcal{I} \in \Pi^*$. This tells the algorithm what M should be before receiving any data.
- An update rule $\delta : \Pi^* \times \Sigma \to \Pi^*$. This tells the algorithm how to update the memory after reading a piece of data.
- A stopping rule $\gamma : \Pi^* \to O$. This tells the algorithm what to output at the end of the execution (after reading all data), depending on the content of the memory (i.e. the value of the variables) at that time. Here O is the set of possible outputs of the algorithm, and this would depend on the task we are trying to solve, and for us O will usually be $\{\mathsf{Accept}, \mathsf{Reject}\}$.

We focus on space usage of the streaming algorithms (and don't card about time usage, at least for now). We now define the space usage.

[1] Here $\lceil x \rceil$ means the smallest integer that is at least x. For example, $\lceil 2.2 \rceil = 3$, $\lceil 2 \rceil = 2$.

Definition 5 (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, the variables of A, over all possible inputs of length at most n.[2]

We now see another example.

Example 6. *Let $L_6 \coloneqq \{w \in \{0,1\}^* \mid$ number of 1's in w is divisible by $4\}$. A streaming algorithm that computes it is as follows:*

- *Variable: a.*
- *Initialization: set $a \coloneqq 0$.*
- *Update rule: on input $\sigma \in \{0,1\}$, if $\sigma = 0$, then let $a \coloneqq a$, otherwise then let*

$$a \coloneqq (a+1) \bmod 4 = \begin{cases} a+1 & a = 0,1,2 \\ 0 & a = 3 \end{cases}.$$

  Formally,

$$\delta(a, \sigma) \coloneqq \begin{cases} a & \sigma = 0 \\ (a+1) \bmod 4 & \sigma = 1 \end{cases}.$$

- *Stopping rule: if $a = 0$, accept, else reject. Formally,*

$$\gamma(a) \coloneqq \begin{cases} \mathsf{Accept} & a = 0 \\ \mathsf{Reject} & a \neq 0 \end{cases}.$$

In the algorithm above, a always keep track of the number of 1's in the portion it has already read modulo 4. Since the possible values of a are $0, 1, 2, 3$, storing a needs $\log_2 4 = 2$ bits, so the space usage of the algorithm is constant.[3]

It is not hard to see that L_6 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. *Every regular language has a streaming algorithm with constant space usage.*

Proof. Since the language is regular, it is recognized by a DFA. Suppose the DFA is $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$. We construct the following streaming algorithm $\mathcal{A}$:

- Variable: $q \in Q$.
- Initialization: set $q \coloneqq q_0$.
- Update rule: set $q \coloneqq \delta(q, \sigma)$ where σ is the current input. Formally, the update rule is exactly δ.
- Stopping rule: if $q \in F$, accept, else reject. Formally,

$$\gamma(q) \coloneqq \begin{cases} \mathsf{Accept} & q \in F \\ \mathsf{Reject} & q \notin F \end{cases}.$$

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

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

We can see that the algorithm $\mathcal{A}$ just simulates the DFA $\mathcal{D}$ on $\mathcal{A}$'s input, and the value of q and the current state of $\mathcal{D}$ is the same after reading the same input. Suppose the final value of q is q^*, since $\mathcal{D}$ recognizes the language, $q \in F$ if the input is in the language and $q \notin F$ otherwise. Therefore, $\mathcal{A}$ accepts if and only if the input is in the language.

Since Q is finite, the space usage of $\mathcal{A}$ is $\log_2 |Q| = O(1)$. □

Theorem 8. *If a language L is computed by a streaming algorithm with constant space usage, then L is regular.*

Proof. Denote by $\mathcal{A}$ the streaming algorithm. Suppose Q is the set of all possible settings of $\mathcal{A}$. Since $\mathcal{A}$ uses constant space, Q is finite. Suppose the initial setting of $\mathcal{A}$ is q_0, its update rule is δ and its stopping rule is γ. We define a DFA $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ as follows.

- The set of state is just Q.
- The transition function is just δ, that is, when the current input symbol is σ and the previous state is q, the current state will be $\delta(q, \sigma)$.
- The start state is just q_0.
- The set of accepting states $F \coloneqq \{q \in Q \mid \gamma(q) = \mathsf{Accept}\}$.

We can see (and formally prove by induction on t) that if the algorithm $\mathcal{A}$ and the DFA $\mathcal{D}$ read the same input $x_1, x_2, \ldots, x_t$, the current setting of the memory of $\mathcal{A}$ will be the same as the current state of $\mathcal{D}$. Suppose the final setting of the memory of $\mathcal{A}$ is q^*, then the final state of $\mathcal{D}$ is also q^*. Therefore, $\mathcal{A}$ accepts if and only if $\gamma(q) = \mathsf{Accept}$, which is equivalent to $q \in F$. Thus, $\mathcal{D}$ recognizes the same language as $\mathcal{A}$, so $\mathcal{D}$ recognizes L and L is regular. □

Now we see yet another example.

Example 9. *Let $L_9 \coloneqq \{w \in \{0,1\}^* \mid w$ is a palindrome$\}$. Here we call a string a palindrome if it reads the same forwards and backwards. (For example,* 111010111 *is a palindrome.) We construct a streaming algorithm for L_9 as follows.*

- *Variable: a.*
- *Initialization: let $a \coloneqq \varepsilon$.*
- *Update rule: on input $\sigma \in \Sigma$, set $a = a \circ \sigma$.*
- *Stop rule: accept if a is a palindrome, otherwise reject.*

We can see that after reading the whole input, a is just the whole input, so the algorithm works. The space usage of the above algorithm is $O(n)$ since we will store the whole input string in a. We will show later that we cannot do better than this.

Besides, we can see that the algorithm above is not specific to this language, except for the stop rule. In fact, for every algorithm there is a similar algorithm.

Theorem 10. *Every language has a streaming algorithm that takes $O(n)$ space.*

Proof. Let L be an arbitrary language. We construct the following algorithm.

- Variable: w.
- Initialization: set $w = \varepsilon$.
- Update rule: on input $\sigma \in \{0, 1\}$, set $w = w \circ \sigma$.
- Stop rule: accept if $w \in L$, reject otherwise.

After reading the whole input, w will become the input string, so the algorithm accepts if and only if $w \in L$. During the execution of the algorithm, w is a string of length at most n, so the algorithm takes $O(n \log |\Sigma|) = O(n)$ space where Σ is the alphabet of L and is finite. □

To summarize, in this lecture we see three types of streaming algorithms in terms of space usage:

- $O(1)$-space streaming algorithm for any regular language.
- $O(\log n)$-space streaming algorithm for L_2 defined in Example 2.
- $O(n)$-space streaming algorithm for any language.