COMS W3261, Section 1, Spring 2024
Streaming Algorithms

Instructor : Josh Alman
Notes by : William Pires

1 Motivations

Streaming algorithms are used in problems where one receives pieces of data on after the other in a sequence $x_1, x_2, \ldots, x_n$ with the goal to compute some statistics / function of the data. For instance if each x_i is a number, the goal could be to compute their average. However, once a piece has been received, you aren't able to see it again [1].

An obvious solution to the issue would be store each piece of data as soon as you receive it, and then perform the task you wanted. However, there might be so much data that it would be impossible (or too expensive) to store all of it. For instance Google Trends is able to tell you what people have recently been searching. Google doesn't store a big list of all the searches done by everyone, that would be way too much storage. They only store some statistics of the data, which is enough to tell how much something has been googled over time.

Hence the goal of a streaming algorithm is to perform a task while using as little space as possible (which hopefully is much less than storing $x_1, \ldots, x_n$).

2 Streaming Algorithm

Definition 1 (Streaming Algorithm)**.** A streaming algorithm $\mathcal{A}$ over the alphabet Σ has a working memory $M \in \Sigma^*$. The memory can be viewed as set of variables $M = \{X_1, \ldots, X_\ell\}$. These variables can store different kind of information, and their number might change during the execution of the algorithm. The algorithm is made of the following three components :

- An initialization rule $\mathcal{I} \in \Sigma^*$. This tells the algorithm what M should be before receiving any data.
- An update rule $\delta : \Sigma^* \times \Sigma \to \Sigma^*$. This tells the algorithm how to update the memory
- A stopping rule $\gamma : \Sigma^* \to O$. Here O is the set of possible "answers" of the algorithm. This would depend on the task you're trying to solve, for us O will always be $\{\mathsf{Accept}, \mathsf{Reject}\}$

[1] In some settings, you are allowed to go over all the data $k \geq 2$ times (this is called the number of passes). But after seeing x_i, you'd have to loop through $x_{i+1}, \ldots, x_n, x_1, \ldots, x_{i-1}$ if you want to see it again.

We introduced some extra notation compared to the one presented in class. This will be needed later to proof the equivalence between streaming algorithms and DFAs formally.

So how does such an algorithm work ? Say the stream is $x = x_1, \ldots, x_n \in \Sigma^n$.

- Initialization : This is before seeing any data, during this stage, the algorithm initializes some variables in M, so that the memory's content becomes $\mathcal{I}$.
- Update : When the algorithm receives $x_i \in \Sigma$, it updates M by replacing it by $\sigma(M, x_i)$. That is, using the update rule, the algorithm updates the variables in memory.
- Stop : After the algorithm has read the last bit of data x_n (and updated M accord to the update rule), it uses the stop rule to output some value. For us it will either be accepting the input or rejecting it.

When it comes to streaming algorithms, we will be interested in the space taken by $\mathcal{A}$. That is given a stream of n elements $x_1, \ldots, x_n$, what is the largest number of bits it takes to write the variables stored in M.

Definition 2 (Space usage)**.** Space usage of streaming algorithm $\mathcal{A}$ is a function $S : \mathbb{N} \to \mathbb{N}$, where $S(n)$ is the max number of bits used to store M the variables in M over all inputs of length at most n.

Definition 3 (Constant space algorithm)**.** Let $\mathcal{A}$ be a streaming algorithm over alphabet Σ. We say that $\mathcal{A}$ is a constant space streaming algorithm if there exists a constant $m \in \mathbb{N}$ such that for all $n \in \mathbb{N}$ we have $S(n) \leq m$.

3 Some examples

We will now go through some examples of streaming algorithms. Given a language L we want to design a streaming algorithm that decides L.

Definition 4. Let Σ be an alphabet, for a language $L \subseteq \Sigma^*$. We say that a streaming algorithm $\mathcal{A}$ decides L, if given input $x_1, \ldots, x_n \in \Sigma^n$, $\mathcal{A}$ outputs Accept if $x_1 x_n \ldots x_n \in L$, and otherwise outputs Reject.

Example 1. Let $L = \{w \in \{0,1\}^* \mid w \text{ has an equal number of 1s and 0s } \}$. We give a streaming algorithm for this task. Here we simply keep track of the number of zeros in Z, and of ones in N and check if they are equal by the end.

- Initialization Rule : Set $Z = 0$, $N = 1$.
- Update Rule : When we see symbol σ :
  - If $\sigma = 0$ set $Z = Z + 1$.
  - Else ($\sigma = 1$) set $N = N + 1$.
- Stop Rule : Output Accept if $Z = N$, otherwise output Reject.

What is the space usage of this algorithm ? We have two variables. On an input of length $\leq n$, both Z and N store an integer in $\{0, \ldots, n\}$. Writing a number up to n takes $\lceil \log_2(n+1) \rceil$ bits at most (storing the number in binary). So in total, we need space at most $S(n) \leq 2\lceil \log_2(n+1) \rceil$ bits. But we can do better !

Consider the following algorithm, where we only keep track of the number of 0s minus the number of 1s.

- Initialization Rule : Set $C = 0$.
- Update Rule : When we see symbol σ,
  - If $\sigma = 0$ set $C = C + 1$.
  - Else ($\sigma = 1$) set $C = C - 1$.
- Stop Rule : Output Accept if $C = 0$, otherwise output Reject.

What is the space usage of this algorithm ? On an input of length at most n, C is always in $\{-n, \ldots, n\}$. Thus storing C takes $\lceil \log_2(2n+1) \rceil$ bits to store. So we can conclude $S(n) \leq \lceil \log_2(2n+1) \rceil$.

For some languages, we can even get streaming algorithms using a constant amount of space. That is, the space doesn't depend on the length of the stream.

Example 2. $L = \{w \in \{0,1\}^* \mid \#$ of 1 in w is divisible by $8\}$. The following algorithm only keeps track of the number of 1 seen so far mod 8.

- Initialization rule : Set $C = 0$.
- Update rule : When we see symbol σ,
  - If $\sigma = 0$ do nothing.
  - Else ($\sigma = 1$) set $C = (C + 1) \mod 8$.
- Stop rule : Output Accept if $C = 0$, otherwise output Reject.

Note that C is always between 0 and 7, so we only need three bits to store C. Thus, $S(n) \leq 3$.

Using the same idea, we get a streaming algorithm for $L' = \{w \in \{0,1\}^* \mid \#$ of 1 in w is divisible by $9\}$.

In which case we'd need 4 bits, since we'd store a number between 0 and 8. [2].

4 Regular Languages and Streaming Algorithms

Note that we can easily get a DFA for $L = \{w \in \{0,1\}^* \mid \#\text{ of 1 in } w \text{ is divisible by } 8\}$. We use the states to record the number of 1 mod 8 in the string. The formal definition of the DFA is as follow :

- State space : $Q := \{q_0, \ldots, q_7\}$
- Alphabet : $\Sigma = \{0, 1\}$
- Start state : q_0
- Accept state : $F := \{q_0\}$
- Transition function δ: $\delta(q_i, 0) = q_i$ and $\delta(q_i, 1) = q_{i+1 \bmod 8}$

This is very similar to our streaming algorithm. In fact, a DFA is a special case of a streaming algorithm, where the memory represent "what is the current state". We have the following theorem:

Theorem 1. If L has a DFA with k states, then there exists a streaming algorithm for L using $\lceil \log_2(k) \rceil$ bits of space.

Proof. Given a DFA $D := \{Q, \Sigma, q_0, F, \delta\}$ where $Q = \{q_0, \ldots, q_k\}$, we consider the following streaming algorithm.

- Initialization rule : Set $V = 0$.
- Update rule : When we see symbol σ, if $q_i = \delta(q, \sigma)$ set $V = i$.
- Stop rule : Output Accept if $q_V \in F$, otherwise output Reject.

It's not hard to see that the algorithm just simulates the DFA, in particular after reading $x_1, x_2 \ldots, x_i$ the memory is $V = j$ if and only if D is in q_j after reading $x_1 x_2 \ldots x_i$. □

But is the opposite true ? If L has a constant space streaming algorithm, then L is regular. The answer is yes, in turns out constant space streaming algorithm are equivalent to DFAs.

[2]The idea generalizes to the language $L_k := \{w \in \{0,1\}^* | \#\text{ of 1 in } w \text{ is divisible by } k\}$. To get a streaming algorithm for this language you'd need $\lceil \log_2(k) \rceil$ bits

Theorem 2. A language L is regular if and only if there exists a constant space streaming algorithm $\mathcal{A}$ that decides L.

Proof. We already proved one direction in Theorem 1, so let's prove the other. Let $L \subseteq \Sigma^*$ be a language that has a constant space streaming algorithm $\mathcal{A} = (\mathcal{I}, \delta, \gamma)$, let $m \in \mathbb{N}$ be the maximum space used by the algorithm on any input.

The idea is that the states of our DFA are all the possible settings of the memory of the algorithm. Since the algorithm always uses $\leq m$ space, we'll need constantly many states. The transition function will mirror the update rule of the algorithm. Here is the formal definition of the DFA :

- State space : $Q := \{q_s \mid s \in \Sigma^*, |s| \leq m\}$
- Alphabet : Σ
- Start state : $q_{\mathcal{I}}$, where $\mathcal{I}$ is the string in the initialization rule of $\mathcal{A}$.
- Accept state : $F := \{q_s \mid \gamma(s) = \mathsf{Accept}\}$, where γ is the stop rule of $\mathcal{A}$.
- Transition function δ': If $\delta(s, \sigma) = s'$ then $\delta'(q_s, \sigma) = s'$. Where δ is the update rule of $\mathcal{A}$.

It's easy to see that that if $\mathcal{A}$ has its memory set to s after seeing $x_1, \ldots, x_n$, then our DFA is in state q_s after seeing the same sequence. Finally the DFA accepts at q_s if and only if $\mathcal{A}$ outputs Accept when it stops and the memory content is s. So our DFA accepts $x_1 \ldots x_n$ if and only if $\mathcal{A}$ outputs Accept on input $x_1, \ldots, x_n$. Since $\mathcal{A}$ decides L, this is a DFA for L. □

Finally, let's look at one last example, where we give a streaming algorithm for a language that isn't regular. By Theorem 2 we know we can't achieve a constant space algorithm.

Example 3. Let $L = \{w' \mid w' = ww \text{ and } w \in \{0,1\}^*\}$. We give the following streaming algorithm. The algorithm simply records all of the input before checking it's in L, thus this algorithm has space $S(n) = n$.

- Initialization rule : Set $w' = \epsilon$
- Update rule : When we see symbol σ, set $w' = w' \circ \sigma$
- Stop rule : Output Accept if w' can be written as ww, otherwise output Reject.

Using a similar idea, we can build a streaming algorithm for any language L over $\Sigma = \{0,1\}$ using at most n space. The algorithm simply remembers all the input x and outputs Accept iff $x \in L$.