CS Theory (Spring '25) February 25, 2025

Lecture Note: Streaming Lower Bounds

Instructor: *Josh Alman*

Recall that last time we learnt that every regular language can be recognized by streaming algorithms using $O(1)$ space, and $O(1)$-space streaming algorithms only recognize regular languages.

We also studied two examples. We showed that

$$L_1 := \{w \in \{0,1\}^* \mid w \text{ has more 0's than 1's}\}$$

has an $O(\log n)$-space streaming algorithm, and

$$L_2 := \{w \in \{0,1\}^* \mid w \text{ is a palindrome}\}$$

has an $O(n)$-space algorithm.

This time, we will prove lower bounds on the space usage of streaming algorithms. In particular, we will prove that there is no possible streaming algorithm for L_1 that uses fewer than $O(\log n)$ space, and there is no possible streaming algorithm for L_2 that uses fewer than $O(n)$ space.

1 Streaming lower bounds

Let's first recall algorithm for L_1:

- Variable: a.
- Initialization: set $a = 0$.
- Update rule: on input $\sigma \in \{0,1\}$, if $\sigma = 0$ then set $a := a + 1$, if $\sigma = 1$ then set $a := a - 1$.
- Stopping rule: if $a > 0$ then accept, else reject.

This streaming algorithm takes $O(\log n)$ space because the variable a can take on values from $-n, -n+1, \ldots, n-1, n$ on an input of length n.

The key idea for proving a lower bound on space usage is to identify some input strings which would give different values of a in our algorithm, and moreover prove that not just our algorithm, but any streaming algorithm for the language must have different memory states for those strings.

Let us pick $\{0, 00, 000, 0000, \ldots, 0^n\}$, and show the following.

Lemma 1. *For any streaming algorithm for L_1 and integers $p \neq q$, the strings 0^p and 0^q must result in different memory states.*

Proof. Without loss of generality, suppose $p < q$. Suppose after reading in either 0^p or 0^q, we then read in 1^p. That is, the whole input string is 0^p1^p or 0^q1^p. If the algorithm was in the same memory state after reading 0^p versus 0^q, then it must also be in the same memory state after reading 0^p1^p versus 0^q1^p.

This is because the update rule depends only on the current memory state and the next symbol we read in. That means the streaming algorithm either accepts both 0^p1^p and 0^q1^p, or rejects both. This is a contradiction since $0^p1^p \notin L_1$, but $0^q1^p \in L_1$ (as we assumed $p < q$). $\square$

We are ready to prove the space usage lower bound with **??**.

Theorem 2. *Any streaming algorithm for L_1 must use at least $\log_2(n)/100$ space for inputs of length up to n.*

Proof. Assume to the contrary we have an algorithm for L_1 that uses less than $(1/100)\log_2(n)$ space.

Consider the set of inputs $S = \{0, 00, 000, 0000, \ldots, 0^n\}$.

Since A uses less than $(1/100)\log_2(n)$ space, the number of possible memory configurations of A is at most[1]

$$\sum_{i=0}^{\log_2(n)/100} 2^i = 2^{\log_2(n)/100+1} - 1 \leq 2n^{1/100}.$$

This is much less than $|S| = n$. Therefore, by the pigeonhole principle, there must be two different strings in S that leads to the same memory configuration of A. This contradicts **??**. $\square$

Below we state the general form of the above method for proving streaming lower bounds.

Definition 3. Fix a language L over alphabet Σ. We say two strings $x, y \in \Sigma^*$ are *distinguishable* if there is a string $z \in \Sigma^*$ such that exactly one of xz and yz is in L.

Definition 4. We call $S_n \subset \Sigma^*$ a *length-n distinguishing set* if

1. all strings in S_n has length at most n;
2. all pairs of strings in S_n are distinguishable.

Theorem 5. *If a language L has a length-n distinguishing set S_n, then any streaming algorithm for L must use at least $(1/100)\log_2|S_n|$ space on inputs of length at most n.*

Proof. The proof is similar to **??**. Assume to the contrary we have an algorithm A for L that uses less than $(1/100)\log_2|S_n|$ space. The number of possible memory configurations of A is thus at most

$$\sum_{i=}^{(1/100)\log_2|S_n|} 2^i = 2^{(1/100)\log_2|S_n|+1} - 1 \leq 2|S_n|^{1/100}.$$

This is much less than $|S_n|$. Therefore, by the pigeonhole principle, there must be two different strings x, y in S_n that lead to the same memory configuration of A. Since S_n is a distinguishing set, x, y is distinguishable. Therefore, there is a string $z \in \Sigma^*$ such that exactly one of xz and yz is in L.

However, since x and y lead to the same memory configuration of A, xz and yz should also lead to the same memory configuration of A (as the update rule depends only on the current memory state

[1] The exact number depends on our model of memory usage. In the model we use, the algorithm can use up to m bits of memory for some m, so if $m = 2$ for example, the memory content can be $\varepsilon, 0, 1, 00, 01, 10$, or 11 and there will be 7 possibilities. If we instead required the streaming algorithm to use exactly m bits of memory, then when $m = 2$ for example, the memory content could be $00, 01, 10$, or 11 and there would be 4 possibilities. However, there will only be a constant factor of different between different models, and this is one of the reasons we use big-O notation: a factor of constant does not matter under big-O notation.

and the next symbol). Therefore, xz and yz are either both accepted by A or both rejected by A. This contradicts that only one of xz and yz is in L. $\square$

Now we use **??** to show the following.

Theorem 6. *Streaming algorithms for* $L_2 :- \{w \in \{0,1\}^n \mid w \text{ is a palindrome}\}$ *need at least* $n/100$ *space.*

Proof. Let $S_n = \{0,1\}^n$. This is a distinguishing set because for any distinct $x, y \in \{0,1\}^n$, x, y can be distinguished with $z = \text{reverse}(x)$, where $\text{reverse}(x)$ is the string x flipped backward (for example, $\text{reverse}(00111) = 11100$). Actually, $xz \in L_2$, while $yz \notin L_2$. By **??**, streaming algorithms for L_2 need at least $(1/100)\log_2 |S_n| = n/100$ space. $\square$

To summarize, today we proved that the space usage of the algorithms we discussed in the last lecture for L_1 and L_2 are optimal (up to constant factor).

?? also has the following corollary. (Recall that every regular language has $O(1)$-space streaming algorithms.)

Corollary 7. *If* L *has superconstant-sized length-*n *distinguishing sets, then* L *is not a regular language.*

Here superconstant means not $O(1)$. Formally, $f(n)$ is superconstant if $\forall c > 0$, $\exists n > 0$ such that $f(n) > c$.