Lecture 7: Streaming Algorithms and Communication Complexity
L is not regular

*if and only if*

There are infinitely many strings \(w_1, w_2, \ldots\) so that for all \(i \neq j\), there’s a string \(z\) such that

*exactly one* of \(w_i z\) and \(w_j z\) is in \(L\)

To prove that \(L\) is regular, we have to show that a special finite object (DFA/NFA/regex) exists.

To prove that \(L\) is not regular, it is sufficient to show that a special infinite set of strings exists!

We can prove the **nonexistence** of a DFA/NFA/regex by proving the **existence** of this special string set!
Streaming Algorithms

Have three components

Initialize:
<variables and their assignments>

When next symbol seen is $\sigma$:
<pseudocode using $\sigma$ and vars>

When stream stops (end of string):
<accept/reject condition on vars>
(or: <pseudocode for output>)

Algorithm A computes $L \subseteq \Sigma^*$ if
A accepts the strings in $L$, rejects strings not in $L$
Streaming Lower Bounds from DFAs

For any $L \subseteq \Sigma^*$ define $L_n = \{ x \in L \mid |x| \leq n \}$

**Theorem:** Suppose $L'$ is computable by a streaming algorithm $A$ using $f(n)$ bits of space, on all strings of length up to $n$. Then there is a DFA with $< 2^{f(n)+1}$ states recognizing $L'_n$

**Corollary:** Suppose all DFAs recognizing $L'_n$ need at least $Q(n)$ states. Then $L'$ is not computable by a streaming algorithm using $\log_2 Q(n)$ bits of space!
L is not regular \textit{if and only if} 
There are infinitely many strings \(w_1, w_2, \ldots\) so that for all \(i \neq j\), there's a string \(z\) such that exactly one of \(w_iz\) and \(w_jz\) is in \(L\).

In fact, Myhill-Nerode shows that the size of a distinguishing set for \(L\) is a \textit{lower bound} on the number of states in a DFA for \(L\).

In other words, if \(S\) is a distinguishing set for \(L\), then any DFA for \(L\) must have at least \(|S|\) states.

We can use this fact to prove lower bounds on streaming algorithms!
$L = \{x \mid x \text{ has more 1's than 0's}\}$

Is there a streaming algorithm for $L$ using much \textit{less than} $\log_2 n$ space?

\textbf{Theorem}: Every streaming algorithm for $L$ needs at least $\log_2 n$ bits of space.

We will use:

- Myhill-Nerode Theorem
- The connection between DFAs and streaming
L = \{x \mid x \text{ has more 1's than 0's}\}

**Theorem:** Every streaming algorithm for L requires at least \((\log_2 n)\) bits of space.

**Proof Idea:** Let \(n\) be even, and \(L_n = \{x \in L \mid |x| \leq n\}\)

We will give a *distinguishing set* \(T_n\) for \(L_n\) such that \(|T_n| = n\)

**By the Myhill-Nerode Theorem**

\(\Rightarrow\) Every DFA for \(L_n\) needs at least \(n\) states

\(\Rightarrow\) Every streaming algorithm for \(L\) needs at least 
\((\log n)\) space on strings of length \(\leq n\)
Let $L = \{ x \mid x \text{ has more 1's than 0's} \}$

**Theorem:** Every streaming algorithm for $L$ requires at least $\log_2 n$ bits of space

**Proof (Slide 1):** Let $T_n = \{ 0^i , 1^i \mid i = 1, \ldots, n/2 \}$

**Claim:** $T_n$ is a *distinguishing set* for $L_n$

**Case 1:** Let $x=0^a$ and $y=1^b$ be any strings from $T_n$

**Claim:** $z = \varepsilon$ distinguishes $x$ and $y$ in $L_n$

- $xz$ has more 0s than 1s $\Rightarrow xz \not\in L_n$
- $yz$ has length $\leq n$ and more 1s than 0s $\Rightarrow yz \in L_n$

So the string $z$ distinguishes $x$ and $y$, and $x \not\equiv_{L_n} y$
L = \{x \mid x \text{ has more 1's than 0's}\}

**Theorem:** Every streaming algorithm for L requires at least \((\log_2 n)\) bits of space

**Proof (Slide 2):** Let \(T_n = \{0^i, 1^i \mid i = 1, \ldots, n/2\}\)

Claim: \(T_n\) is a *distinguishing set* for \(L_n\)

Case 2: Let \(x=0^a\) and \(y=0^b\) be from \(T_n\), with \(a < b\)

Claim: \(z = 1^b\) distinguishes \(x\) and \(y\) in \(L_n\)

\[xz\] has length \(\leq n\), \(a\) 0's and \(b\) 1's \(\Rightarrow xz \in L_n\)

\[yz\] has \(b\) zeroes and \(b\) ones \(\Rightarrow yz \notin L_n\)

So the string \(z\) distinguishes \(x\) and \(y\), and \(x \not\equiv_{L_n} y\)
L = \{x \mid x \text{ has more 1's than 0's}\}

**Theorem:** Every streaming algorithm for L requires at least \((\log_2 n)\) bits of space

**Proof (Slide 3):** Let \(T_n = \{0^i, 1^i \mid i = 1, ..., n/2\}\)

**Claim:** \(T_n\) is a *distinguishing set* for \(L_n\)

**Case 3:** Let \(x=1^a\) and \(y=1^b\) be from \(T_n\), with \(a < b\)

**Claim:** \(z = 0^a\) distinguishes \(x\) and \(y\) in \(L_n\)

\(xz\) has \(a\) ones and \(a\) zeroes \(\Rightarrow xz \not\in L_n\)

\(yz\) has length \(\leq n\), \(a\) zeroes and \(b\) ones \(\Rightarrow yz \in L_n\)

So the string \(z\) distinguishes \(x\) and \(y\), and \(x \not\equiv_{L_n} y\)
\( L = \{ x \mid x \text{ has more } 1\text{'s than } 0\text{'s} \} \)

**Theorem:** Every streaming algorithm for \( L \) requires at least \((\log_2 n)\) bits of space.

**Proof (Slide 4):** Let \( T_n = \{0^i, 1^i \mid i = 1, \ldots, n/2\} \)

All pairs of strings in \( T_n \) are distinguishable to \( L_n \)

\[ \implies \text{There are at least } |T_n| \text{ equiv classes of } \equiv_{L_n} \]

By the **Myhill-Nerode Theorem**:

\[ \implies \text{All DFAs recognizing } L_n \text{ need } \geq |T_n| \text{ states} \]

\[ \implies \text{Every streaming algorithm for } L \text{ needs at least } S(n) = (\log_2 |T_n|) \text{ bits of space.} \]

Finally, note \(|T_n| = n\) and we’re done!
A streaming algorithm for
L = \{x \mid x \text{ has more 1's than 0's}\}
tells us if 1's occur more frequently than 0's.

What if the alphabet is more than just 1's and 0's?

And what if we want to find the “top 10” symbols?

FREQUENT ITEMS: Given k and a string \( x = x_1 \ldots x_n \in \Sigma^n \),
output the set \( S = \{\sigma \in \Sigma \mid \sigma \text{ occurs } > n/k \text{ times in } x\} \)

(Question: How large can the set S be?)
FREQUENT ITEMS: Given $k$ and a string $x = x_1 \ldots x_n \in \Sigma^n$, output the set $S = \{\sigma \in \Sigma \mid \sigma \text{ occurs } > n/k \text{ times in } x\}$

Theorem: There is a two-pass streaming algorithm for FREQUENT ITEMS using $(k-1)(\log |\Sigma| + \log n)$ space.

1st pass: Initialize a set $T \subseteq \Sigma \times \{1,\ldots,n\}$ (originally empty)
When the next symbol $\sigma$ is read:
If $(\sigma,m) \in T$, then $T := T - \{(\sigma,m)\} + \{(\sigma,m+1)\}$
Else if $|T| < k-1$ then $T := T + \{(\sigma,1)\}$
Else for all $(\sigma',m') \in T$,
\[ T := T - \{(\sigma',m')\} + \{(\sigma',m'-1)\} \]
If $m' = 0$ then $T := T - \{(\sigma',m')\}$

Claim: At end, $T$ contains all $\sigma$ occurring $> n/k$ times in $x$

2nd pass: Count occurrences of all $\sigma'$ appearing in $T$ to determine those occurring $> n/k$ times
Claim: At end, T contains all $\sigma$ occurring $> n/k$ times in $x$

Proof: By contradiction.
Assume $\sigma'$ occurs $> n/k$ times in $x$ but is never added to $T$.

Obs: Every “k-1 counter decrement” event when reading $\sigma'$ corresponds to $k$ distinct symbols in the stream!

1st pass: Initialize a set $T \subseteq \Sigma \times \{1, \ldots, n\}$ (originally empty)

When the next symbol $\sigma$ is read:

If $(\sigma, m) \in T$, then $T := T - \{(\sigma, m)\} + \{(\sigma, m+1)\}$

Else if $|T| < k-1$ then $T := T + \{(\sigma, 1)\}$

Else for all $(\sigma', m') \in T$,

$T := T - \{(\sigma', m')\} + \{(\sigma', m' - 1)\}$

If $m' = 0$ then $T := T - \{(\sigma', m')\}$

2nd pass: Count occurrences of all $\sigma'$ appearing in $T$
to determine those occurring $> n/k$ times
Number of Distinct Elements

Distinct Elements (DE):
Input: \( x \in \{1, \ldots, 2^k\}^* \), \( n = |x| < 2^{k/2} \)
Output: The *number* of different elements appearing in \( x \); call this \( \text{DE}(x) \)

**Easy:** There is a streaming algorithm for \( \text{DE} \) using \( O(k \cdot n) \) space

**Theorem:** Every streaming algorithm for \( \text{DE} \) requires \( \Omega(k \cdot n) \) space!
Theorem: Every streaming algorithm for DE requires $\Omega(kn)$ space

Let $\Sigma = \{1, \ldots, 2^k\}$. Say $x, y \in \Sigma^*$ are 
**DE distinguishable** if $(\exists z \in \Sigma^*)[\text{DE}(xz) \neq \text{DE}(yz)]$

Lemma: Let $S \subseteq \Sigma^*$ be such that every pair in $S$ is DE distinguishable. Then every streaming algorithm for DE needs $\geq \log_2 |S|$ bits of space.

Proof: Let algorithm A use $< \log_2 |S|$ space. We will show A cannot compute DE on all inputs. By pigeonhole principle, there are distinct $x, y$ in $S$ that lead A to the *same memory state*. So A gives the *same output* on both $xz$ and $yz$. But $\text{DE}(xz) \neq \text{DE}(yz)$, so A does not compute DE.
Lemma: Let $S \subseteq \Sigma^*$ be such that every pair in $S$ is DE distinguishable. Then every streaming algorithm for DE needs $\geq \log_2 |S|$ bits of space.

Lemma: There is a DE distinguishable $S$ of size $\geq 2^{kn/4}$

Proof: For every subset $T$ of $\Sigma$ of size $n/2$, define $x_T$ to be any concatenation of the symbols in $T$.

For distinct sets $T$ and $T'$, $x_T$ and $x_T'$ are distinguishable:
- $x_T x_T$ contains exactly $n/2$ distinct elements
- $x_T' x_T$ has more than $n/2$ distinct elements

The total number of such subsets $T$ is
\[
\binom{2^k}{n/2} \geq 2^{kn/2} / (n/2)^{n/2} \geq 2^{kn/4}, \text{ for } n < 2^{k/2}
\]
Theorem: Every streaming algorithm for DE requires $\Omega(kn)$ space.

The total number of such subsets is $2^{\Omega(kn)}$, for $2^k > n^2$.

What’s the number of subsets of $\{1, \ldots, 2^k\}$ of size $n/2$?

$$\binom{2^k}{n/2}$$

Want to estimate this quantity. Use $\left(\frac{a}{b}\right)^b \geq \left(\frac{a}{b}\right)^b$.

Then

$$\binom{2^k}{n/2} \geq \left(\frac{2^k}{n/2}\right)^{n/2} \geq \frac{2^{kn}}{2^{n/4}}.$$  

Since

$$\left(\frac{n}{2}\right)^{n/2} \left(\frac{2^k}{n/2}\right)^{n/2} \left(\frac{2^2 k}{2}\right)^{n/4} \left(\frac{2}{n/2}\right)^{n/2} \left(\frac{k}{n/2}\right)^{kn/4} > \frac{2^{kn}}{2^{n/4}},$$  

we have

$$\left(\frac{n}{2}\right)^{n/2} \left(\frac{2^k}{n/2}\right)^{n/2} \left(\frac{2^2 k}{2}\right)^{n/4} \left(\frac{2}{n/2}\right)^{n/2} \left(\frac{k}{n/2}\right)^{kn/4} > \frac{2^{kn}}{2^{n/4}}.$$
Theorem: Every streaming algorithm for approximating the number of DE to within ± 20% error also requires $\Omega(k n)$ space!

See Lecture Notes.
Distinct Elements (DE)
Input: $x \in \{1, \ldots, 2^k\}^*$, $n = |x| < 2^{k/2}$
Output: The number of distinct elements appearing in $x$

**Theorem:** There is a *randomized* streaming algorithm that w.h.p. approximates DE to within 0.1% error, using $O(k + \log n)$ space!

**Recall:** *Deterministic* streaming algorithms require at least $\Omega(kn)$ space.
Randomized Algorithm for DE

Idea: Let $h: \{1, \ldots, 2^k\} \to [0, 1]$ be a random function. 
(For all $i \in \{1, \ldots, 2^k\}$, pick $j \in \{1, \ldots, n^2\}$ at random, $h(i) := j/n^2$)

Initialize $m := 1$.
When $x_i$ is read, update $m := \min\{m, h(x_i)\}$.
At the end of the stream, return $1/m$.

Obs: $m = \text{minimum of DE}(x)$ random numbers in $[0,1]$

Claim: Let $x \in \{1, \ldots, 2^k\}^*$
With probability $> 0.8$, $\text{DE}(x)/5 \leq 1/m \leq 10 \cdot \text{DE}(x)$.

*Can boost accuracy using $O(1)$ more hash functions!*
(See the Lecture Notes!)
Initialize $m := 1$. When $x_i$ is read, update $m := \min\{m, h(x_i)\}$. At the end of the stream, return $1/m$.

Claim: With prob. $> 0.8$, $1/m$ is between $DE(x)/5$ and $10 \cdot DE(x)$.

Assuming $h$ is random, $m$ equals the minimum of $DE(x)$ random numbers in $[0, 1]$. Let $L = DE(x)$.

By union bound,

\[
\Pr[\min \text{ of } L \text{ random numbers in } [0,1] < \frac{1}{10L}] \leq L \cdot \Pr[\text{a random number in } [0,1] \text{ is } < \frac{1}{10L}] = \frac{1}{10}.
\]

\[
\Pr[\min \text{ of } L \text{ random numbers in } [0,1] \geq \frac{5}{L}] = (1-\frac{5}{L})^L = [(1-\frac{5}{L})^{L/5}]^5 \leq 1/e^5 < 0.007.
\]

Therefore, $\Pr[1/m \text{ is between } L/5 \text{ and } 10L] > 0.893$. 

\[
(1-\frac{1}{x})^x \leq 1/e
\]
Randomized Algorithm for DE

Idea: Let $h: \{1, \ldots, 2^k\} \to [0, 1]$ be a random function. (For all $i \in \{1, \ldots, 2^k\}$, pick $j \in \{1, \ldots, n^2\}$ at random, $h(i) := j/n^2$)

Initialize $m := 1$.
When $x_i$ is read, update $m := \min\{m, h(x_i)\}$.
At the end of the stream, return $1/m$.

Naively, this uses $2^k \log(n)$ space to store $h$!
Use special (pairwise-independent) hash functions which can be stored with only $O(k + \log(n))$ space.
Communication Complexity
Communication Complexity

A theoretical model of distributed computing

- **Function** \( f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\} \)
  - Two inputs, \( x \in \{0,1\}^* \) and \( y \in \{0,1\}^* \)
  - We assume \( |x| = |y| = n \). Think of \( n \) as HUGE

- **Two computers**: Alice and Bob
  - Alice *only* knows \( x \), Bob *only* knows \( y \)

- **Goal**: Compute \( f(x, y) \) by communicating as few bits as possible between Alice and Bob

*We do not count computation cost.* We only care about the number of bits communicated.
In every step: A bit or STOP is sent, which is a function of the party's input and all the bits communicated so far.

Communication cost = number of bits communicated
= 4 (in the example)

We assume Alice and Bob alternate in communicating, and the last BIT sent is the value of $f(x,y)$.
Def. A protocol for a function \( f \) is a pair of functions \( A, B : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0, 1, \text{STOP}\} \) with the semantics:

On input \((x, y)\), let \( r := 0, b_0 = \epsilon \).

While \((b_r \neq \text{STOP})\),

\[ r++ \]

If \( r \) is odd, Alice sends \( b_r = A(x, b_1 \cdots b_{r-1}) \)
else Bob sends \( b_r = B(y, b_1 \cdots b_{r-1}) \)

Output \( b_{r-1} \). Number of rounds \( = r - 1 \)
Def. The *cost of a protocol* \( P \) for \( f \) on \( n \)-bit strings is

\[
\max_{x,y \in \{0,1\}^n} \text{[number of rounds in } P \text{ to compute } f(x, y)]
\]

The *communication complexity* of \( f \) on \( n \)-bit strings, \( cc(f) \), is *minimum cost* over *all protocols* for \( f \) on \( n \)-bit strings

\( = \) the minimum number of rounds used by any protocol computing \( f(x, y) \), over all \( n \)-bit \( x, y \)
Example. Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be arbitrary

There is always a “trivial” protocol:

Alice sends the bits of her $x$ in odd rounds
Bob sends whatever bit he wants in even rounds
After $2n - 1$ rounds, Bob knows $x$ and can send $f(x, y)$

Proposition: For every $f$, $cc(f) \leq 2n$
Example. \( \text{PARITY}(x, y) = \sum_i x_i + \sum_i y_i \mod 2.\)

What’s a good protocol for computing PARITY?

Alice sends \( b_1 = (\sum_i x_i \mod 2) \)
Bob sends \( b_2 = (b_1 + \sum_i y_i \mod 2). \) Alice stops.

Proposition: \( \text{cc}(\text{PARITY}) = 2 \)
Example. $\text{MAJORITY}(x, y) = \text{most frequent bit in } xy$

What's a good protocol for computing $\text{MAJORITY}$?

Alice sends $N_x = \text{number of 1s in } x$
Bob computes $N_y = \text{number of 1s in } y$,
sends 1 iff $N_x + N_y$ is greater than $(|x| + |y|)/2 = n$

**Proposition:** $cc(\text{MAJORITY}) \leq O(\log n)$
Example. \( \text{EQUALS}(x, y) = 1 \iff x = y \)

What’s a good protocol for computing \( \text{EQUALS} \)?

\[??\]

*Communication complexity* of \( \text{EQUALS} \) is at most \( 2n \)
Connection to Streaming and DFAs

Let $L \subseteq \{0,1\}^*$

Def. $f_L: \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ for $x, y$ with $|x| = |y|$ as:

\[
f_L(x, y) = 1 \iff xy \in L
\]

Examples:

$L = \{ x \mid x \text{ has an odd number of } 1\text{s}\}$

\[
\Rightarrow f_L(x, y) = \text{PARITY}(x, y) = \sum_i x_i + \sum_i y_i \mod 2
\]

$L = \{ x \mid x \text{ has more } 1\text{s than } 0\text{s}\}$

\[
\Rightarrow f_L(x, y) = \text{MAJORITY}(x, y)
\]

$L = \{ xx \mid x \in \{0,1\}^*\}$

\[
\Rightarrow f_L(x, y) = \text{EQUALS}(x, y)
\]