Streaming Algorithms
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
DFAs and Streaming

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 for all $n$, there is a DFA $M$ with $< 2^{f(n)+1}$ states such that $L'_n = L(M)_n$

**Corollary:** Suppose for all $n$, every DFA $M$ with $L'_n = L(M)_n$ needs $\geq 2^{f(n)+1}$ states. Then $L'$ is *not computable* by a streaming algorithm that uses $f(n)$ bits of space!
L = \{x \mid x \text{ has more } 1\text{'s than } 0\text{'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 \((\log_2 n)\)-2 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 \((\log_2 n) - 2\) bits of space

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

We will give a set \(S_n\) of \(n/2 + 1\) strings such that each pair in \(S_n\) is distinguishable to \(L_n\)

**Myhill-Nerode Thm** \(\Rightarrow\) Every DFA recognizing \(L_n\) needs at least \(n/2 + 1\) states

\(\Rightarrow\) Every streaming algorithm for L needs at least \((\log n) - 2\) bits of memory on strings of length n
**L = {x | x has more 1’s than 0’s}**

**Theorem:** Every streaming algorithm for L requires at least \((\log_2 n) - 2\) bits of space.

Suppose we partition all strings into their equivalence classes under \(\equiv_{L_n}\).

Construct \(S_n\)

But the number of states in a DFA recognizing \(L_n\) is **at least** the number of equivalence classes under \(\equiv_{L_n}\).
L = \{x \mid x \text{ has more 1's than 0's}\}

**Theorem:** Every streaming algorithm for L requires at least \((\log_2 n)-2\) bits of space

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

Let \(x = 0^{n/2-k}1^k\) and \(y = 0^{n/2-j}1^j\) be from \(S_n\), with \(k > j\)

**Claim:** \(z = 0^{k-1}1^{n/2-(k-1)}\) distinguishes \(x\) and \(y\) in \(L_n\)

\(xz\) has \(n/2-1\) zeroes and \(n/2+1\) ones \(\Rightarrow xz \in L_n\)

\(yz\) has \(n/2+(k-j-1)\) zeroes and \(n/2-(k-j-1)\) ones

But \(k-j-1 \geq 0\) \(\ldots\) so \(yz \notin L_n\)

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 \((\log_2 n) - 2\) bits of space

**Proof (Slide 2):**

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

\(\Rightarrow\) There are at least \(|S_n|\) equiv classes of \(\equiv_{L_n}\)

By the Myhill-Nerode Theorem:

\(\Rightarrow\) All DFAs recognizing \(L_n\) need \(\geq |S_n|\) states

\(\Rightarrow\) There is no streaming algorithm for L using \(f(n) = (\log_2 |S_n|) - 1\) bits of space.

Recall \(|S_n| = n/2 + 1\) ... and we’re done!
“heavy hitters”

Finding Frequent Items

A streaming algorithm for
L = \{x \mid x \text{ has more } 1\text{'s than } 0\text{'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 an 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
Number of Distinct Elements

The DE problem
Input: \( x \in \{0,1,\ldots,2^k-1\}^* \), \( n=|x| < 2^{k/2} \)
Output: The number of distinct elements appearing in \( x \)

Note: There is a streaming algorithm for DE using \( O(k \ n) \) space

Theorem: Every streaming algorithm for DE requires \( \Omega(k \ n) \) space
Theorem: Every streaming algorithm for DE requires $\Omega(kn)$ space.

Let $\Sigma = \{0, 1, ..., 2^k-1\}$

Define: $x, y \in \Sigma^*$ are **DE distinguishable** if

$(\exists \ z \in \Sigma^*)[xz \text{ and } yz \text{ contain a different number of distinct elements}]$

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: *Pigeonhole Principle!* If an algorithm $A$ uses $< (\log_2 |S|)$ bits, there are distinct $x, y$ in $S$ that lead $A$ to the same memory state. Consider $xz$ and $yz$...
Theorem: Every streaming algorithm for DE requires $\Omega(k n)$ space

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 $2^{\Omega(k n)}$

Proof: For each 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 is $2^{\Omega(k n)}$, for $n < 2^{k/2}$
Theorem: Every streaming algorithm for DE requires $\Omega(k \cdot n)$ space.

The total number of such subsets is $2^{\Omega(k \cdot n)}$, 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 $\left(\frac{a}{b}\right)^b \geq \frac{a}{b}$

Then

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

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

See Lecture Notes.
Randomized Algorithms Help!

The DE problem
Input: \( x \in \{0,1,...,2^k\}^* \), \( n = |x| < 2^{k/2} \)
Output: The number of distinct elements appearing in \( x \)

Theorem: There is a randomized streaming algorithm that can approximate DE to within 0.1% error, using \( O(k + \log n) \) space!
Randomized Algorithm for DE

Define \( h: \{0,1,\ldots,2^k-1\} \rightarrow [0, 1] \)
to be a random hash function.

Store a value \( m \), initialized at 1.
For each \( i \), see \( x_i \) and update \( m \leftarrow \min\{m, h(x_i)\} \).
At the end of the stream, return \( \frac{1}{m} \).

Claim: Let \( L \) be the number of DE.
With probability \( > 0.8 \), \( \frac{1}{m} \) is between \( \frac{L}{5} \) and \( 10L \).
Store a value m, initialized at 1.
For each i, see $x_i$ and update $m \leftarrow \min\{m, h(x_i)\}$.
At the end of the stream, return $1/m$.

If h is random, then the algorithm selects m as the minimum of L random numbers in [0, 1].

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

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

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

Can boost the accuracy by using more hash functions.
Randomized Algorithm for DE

Store a value \( m \), initialized at 1.
For each \( i \), see \( x_i \) and update \( m \leftarrow \min\{m, h(x_i)\} \).
At the end of the stream, return \( \frac{1}{m} \).

With probability \( > 0.89 \) alg. returns a constant approximation to the number of DE.

Space usage: store \( h \), store \( m \).

\( h \) – not fully random, but pairwise independent
Can store \( h \) in \( O(k) \) space.
Use bounded precision reals: precision \( O(\log n) \) suffices
Randomized Algorithms Help!

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

Recall: *Deterministic* streaming algorithms require $\Omega(kn)$ 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.
Alice and Bob Have a Conversation

In every step: A bit 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 $r := 0$, $b_0 = \varepsilon$.

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 is the minimum cost over all protocols for $f$ on $n$-bit strings = the minimum number of rounds used by any protocol that computes $f(x, y)$, over all $n$-bit $x, y$. 

\[ f(x, y) = 0 \]
\[ A(x, \varepsilon) = 0 \]
\[ B(y, 0) = 1 \]
\[ A(x, 01) = 1 \]
\[ B(y, 011) = 0 \]
Example. Let $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ be arbitrary.

There is always a “trivial” protocol:

Alice sends the bits of her $x$ in odd rounds
Bob sends the bits of his $y$ in even rounds

After $2n$ rounds, they both know each other’s input!

*The communication complexity of every $f$ is at most $2n$*
Example. 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.

The communication complexity of PARITY is 2
Example. $\text{MAJORITY}(x, y) = \text{most frequent bit in } xy$

What’s a good protocol for computing 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 $(|x|+|y|)/2 = n$

Communication complexity of MAJORITY is $O(\log n)$
Example. $\text{EQUALS}(x, y) = 1 \iff x = y$

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

$\textbf{????}$

*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:

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

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

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

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

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

$$\Rightarrow f_L(x, y) = \text{EQUALS}(x, y)$$
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:

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

Theorem: If $L$ has a streaming algorithm using $\leq s$ space, then the comm. complexity of $f_L$ is at most $4s + 5$.

Proof: Alice runs streaming algorithm $A$ on $x$.
Sends the memory content of $A$: this is $s$ bits of space
Bob starts up $A$ with that memory content, runs $A$ on $y$.
Gets an output bit, sends to Alice.

(...why $4s+5$ rounds? Can you do better?)