Homework 1 is due!

Homework 2 coming soon
CS 154, Lecture 5:
The Myhill-Nerode Theorem,
On Learning DFAs
Streaming Algorithms
The Myhill-Nerode Theorem
In DFA Minimization, we defined an equivalence relation between states. We can also define a similar equivalence relation over strings and languages:

Let $L \subseteq \Sigma^*$ and $x, y \in \Sigma^*$

$x \equiv_L y$ if for all $z \in \Sigma^*$, $xz \in L \iff yz \in L$

Define: $x$ and $y$ are indistinguishable to $L$ if $x \equiv_L y$

Claim: $\equiv_L$ is an equivalence relation

Proof?
Let \( L \subseteq \Sigma^* \) and \( x, y \in \Sigma^* \)
\[ x \equiv_L y \text{ if for all } z \in \Sigma^*, [xz \in L \iff yz \in L] \]

The Myhill-Nerode Theorem:
A language \( L \) is regular if and only if
the number of equivalence classes of \( \equiv_L \) is finite.

Proof (\( \Rightarrow \)) Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a min DFA for \( L \).
Define the relation: \( x \sim_M y \) if \( \Delta(q_0, x) = \Delta(q_0, y) \)
Claim: \( \sim_M \) is an equivalence relation with \( |Q| \) classes
Claim: If \( x \sim_M y \) then \( x \equiv_L y \)
Proof: \( x \sim_M y \) implies for all \( z \in \Sigma^* \), \( xz \) and \( yz \) reach the same state of \( M \).
So \( xz \in L \iff yz \in L \), and \( x \equiv_L y \)
Corollary: Number of equivalence classes of \( \equiv_L \) is at most the number of equivalence classes of \( \sim_M \) (which is \( |Q| \))
Let \( L \subseteq \Sigma^* \) and \( x, y \in \Sigma^* \)
\[ x \equiv_L y \text{ if for all } z \in \Sigma^*, [xz \in L \iff yz \in L] \]

(\( \Leftarrow \)) If the number of equivalence classes of \( \equiv_L \) is \( k \)
then there is a DFA for \( L \) with \( k \) states

**Idea:** Build a DFA using equivalence classes of \( \equiv_L \)

Define a DFA \( M \) where

- \( Q \) is the set of equivalence classes of \( \equiv_L \)
- \( q_0 = [\varepsilon] = \{ y \mid y \equiv_L \varepsilon \} \)
- \( \delta([x], \sigma) = [x \sigma] \)
- \( F = \{ [x] \mid x \in L \} \)

**Claim:** \( M \) accepts \( x \) if and only if \( x \in L \)
The Myhill-Nerode Theorem gives us a new way to prove that a given language is not regular:

**L is not regular if and only if** there are infinitely many equivalence classes of \( \equiv_L \)

\[
L \text{ is not regular if and only if } \exists \text{ infinitely many strings } w_1, w_2, \ldots \text{ so that for all } w_i \neq w_j, w_i \text{ and } w_j \text{ are distinguishable to } L: \\
\text{there is a } z \in \Sigma^* \text{ such that exactly one of } w_iz \text{ and } w_jz \text{ is in } L
\]
The Myhill-Nerode Theorem gives us a new way to prove that a given language is not regular:

**Theorem:** \( L = \{0^n 1^n \mid n \geq 0\} \) is not regular.

**Proof:** Consider the infinite set of strings \( S = \{0, 00, 000, \ldots, 0^n, \ldots\} \)

Take any pair \((0^m, 0^n)\) of distinct strings in \( S \)

Let \( z = 1^m \)

Then \( 0^m 1^m \) is in \( L \), but \( 0^n 1^m \) is not in \( L \)

That is, all pairs of strings in \( S \) are distinguishable

Hence there are infinitely many equivalence classes of \( \equiv_L \), and \( L \) is not regular.
PAC Learning

Concept: $c \in C$

Hypothesis: $h \in H$
Probably Approximately Correct Learning

Instance space: $X$
Concept class: $C$ (functions over $X$)
Hypothesis class $H$ (functions over $X$)

Proper learning $H = C$

Algorithm $A$ PAC-learns $C$ (distribution free) if:
$\forall c \in C$ and $\forall$ distribution $D$ over $X$,
$A$ gets as input $(x_1, c(x_1)), \ldots, (x_m, c(x_m))$, where $x_i$ distributed according to $D$ and outputs $h \in H$ Such that

$$\Pr_A[ \Pr_{x \in D}[h(x) \neq c(x)] > \delta ] < \varepsilon$$
Occam's Razor

Many formulations (before and after Ockham 1287–1347): “We consider it a good principle to explain the phenomena by the simplest hypothesis possible” Ptolemy (c. AD 90 – c. AD 168)

One of several technical interpretation (loosely put):

Any algorithm $A$ that outputs a “small” hypothesis $h$ that explains its input $(x_1, c(x_1)), \ldots, (x_m, c(x_m))$, then $A$ satisfies the definition of PAC learning (it generalizes)

Why?
PAC-Learning DFA

Given a regular language \( L \), and given examples \( w_1 \ldots w_m \) (positive and negative), learn a small DFA that is consistent with \( L \) on these inputs.

One approach: enumerate all DFAs (from smallest) till you find one that is consistent with input.

Exponential in size of DFA. Can we do better?
PAC-Learning DFA

Given a regular language \( L \), and given examples \( w_1 \ldots w_m \) (positive and negative), learn a DFA that is consistent with \( L \) on these inputs.

Define \( L^? \) to be the corresponding “partially defined language” such that either \( w \in L^? \) or \( w \not\in L^? \) or we don’t know.

\( L^? \) distinguishes \( x \) and \( y \) if there exist \( z \) such that either \( xz \in L^? \) and \( yz \not\in L^? \) or vice versa.

Claim: if \( x \equiv_L y \) then \( x \) and \( y \) cannot be distinguished by \( L^? \)

Use similar approach to the proof of Myhill-Nerode ??
PAC-Learning DFA

Given a regular language $L$, and given examples $w_1 \ldots w_m$ (positive and negative), learn a small DFA that is consistent with $L$ on these inputs.

Enumerating all DFAs - exponential in size of DFA. Can we do better?

Hardness results – probably not!

Can learn automata actively!
Related to Myhill-Nerode but set of examples has to be carefully designed: PAC-learning with membership queries (+ other variants)
Streaming Algorithms
Streaming Algorithms

Here: vague on computation cost (less of an issue for DFAs).
All our examples – efficient computation
\[ L = \{x \mid x \text{ has more 1's than 0's}\} \]

Initialize: \( C := 0 \) and \( B := 0 \)

When the next symbol \( \sigma \) is read,

If \( (C = 0) \) then \( B := \sigma, C := 1 \)

If \( (C \neq 0) \) and \( (B = \sigma) \) then \( C := C + 1 \)

If \( (C \neq 0) \) and \( (B \neq \sigma) \) then \( C := C - 1 \)

When the stream stops,

\( \text{accept if } B=1 \) and \( C > 0 \), else \( \text{reject} \)

\( B = \) the majority bit

\( C = \) how many more times that \( B \) appears

On all strings of length \( n \), the algorithm uses \( (1 + \log_2 n) \) bits of space \( (\text{to store } B \text{ and } C) \)
Streaming algorithms differ from DFAs in several significant ways:

1. Streaming algorithms can output more than one bit

2. The “memory” or “space” of a streaming algorithm can (slowly) increase as it reads longer strings

3. Sometimes allow making multiple passes over the data; Could be randomized
Theorem: Suppose a language $L$ can be recognized by a DFA $M$ with $\leq 2^p$ states. Then $L$ is computable by a streaming algorithm $A$ using $\leq p$ bits of space.

Proof Idea: Can define algorithm $A$ as follows:
Initialize: Encode the start state of $M$ in memory.
When the next symbol $\sigma$ is read: using the transition function of $M$, update the state of $M$.
When the string ends: Output accept if the current state of $M$ is a final state, reject otherwise.
DFAs and Streaming

∀L ⊆ Σ* define \( L_n = \{ x \in L \mid |x|=n \} \); \( L_{\leq n} = \{ x \in L \mid |x|\leq n \} \)

Theorem: Suppose \( L \) is computable by a streaming algorithm \( A \) using \( f(n) \) bits of space, on all strings of length up to \( n \).
⇒ for all \( n \), there is a DFA \( M \) with \( \leq 2^{f(n)} \) states such that \( L_{\leq n} = L(M)_{\leq n} \)

Proof Idea:
States of \( M = 2^{f(n)} \) possible settings of \( A \)'s memory, on strings of length up to \( n \)
Start state of \( M \) = Initial memory configuration of \( A \)
Transition function = Mimic how \( A \) updates its memory
Accept states of \( M \) = Configurations in which \( A \) would accept, if string ended
Example: $L = \{x \mid x \text{ has more 1's than 0's}\}$

Initialize: $C := 0$ and $B := 0$
When the next symbol $x$ is read,
If $(C = 0)$ then $B := x$, $C := 1$
If $(C \neq 0)$ and $(B = x)$ then $C := C + 1$
If $(C \neq 0)$ and $(B \neq x)$ then $C := C - 1$
When the stream stops,
accept if $B=1$ and $C > 0$, else reject

Want: A DFA that agrees with $L$ on all strings of length $\leq 2$
L = \{x \mid x \text{ has more 1’s than 0’s}\}

Is there a streaming algorithm for L using much less than \((\log_2 n)\) space?

Theorem: Every streaming algorithm for L needs at least \((\log_2 n) - 1\) 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 \((\log_2 n) - 1\) bits of space

Proof Idea: Let \(n\) be even, and \(L_n = \{0,1\}^n \cap L\)
We will give a set \(S_n\) of \(n/2 + 1\) strings such that each pair in \(S_n\) is distinguishable in \(L_n\)

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 \((\log n) - 1\) bits of memory on strings of length \(n\)
$L = \{x \mid x \text{ has more } 1\text{'s than } 0\text{'s}\}$

**Theorem:** Every streaming algorithm for $L$ requires at least $(\log_2 n) - 1$ bits of space.

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

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

**Proof:** 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 \( k-j-1 \geq 0 \) ... so \( yz \notin L_n \)

So the string \( z \) distinguishes \( x \) and \( y \), and \( x \not\in L_n \) \( y \)

**Theorem:** Every streaming algorithm for \( L \) requires at least \((\log_2 n)-1\) bits of space
L = \{x \mid x \text{ has more } 1\text{'s than } 0\text{'s}\}

Theorem: Every streaming algorithm for L requires at least \((\log_2 n) - 1\) bits of space

Proof:
All pairs of strings in \(S_n\) are distinguishable in \(L_n\)
⇒ There are at least \(|S_n|\) equiv classes of \(\equiv_{L_n}\)
   By the Myhill-Nerode Theorem:
   ⇒ All DFAs recognizing \(L_n\) need \(\geq |S_n|\) states
   ⇒ Every streaming algorithm for L requires at least \((\log_2 |S_n|)\) bits of space.
Recall \(|S_n| = n/2 + 1\) and we’re done!
Finding Frequent Items

A streaming algorithm for recognizing 
$L = \{x \mid x \text{ has more 1's than 0's}\}$
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 $S = \{\sigma \in \Sigma \mid \sigma \text{ occurs } > n/k \text{ times in } x\}$

*(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 \( S = \{ \sigma \in \Sigma \mid \sigma \text{ occurs } > \frac{n}{k} \text{ times in } x \} \)

Theorem: There is a two-pass streaming algorithm for Frequent Items using \( O(k (\log |\Sigma| + \log n)) \) space.

1st pass: Initialize a set \( T \subseteq \Sigma \times \mathbb{N} \) (originally empty)
Read the next symbol \( \sigma \) from the stream.
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: \( T \) contains all \( \sigma \) occurring \( > \frac{n}{k} \) times in \( x \)

2nd pass: Count occurrences of all \( \sigma' \) appearing in \( T \)
to determine those occurring \( > \frac{n}{k} \) times.
Number of Distinct Elements

The DE problem
Input: \( x \in \{0,1,...,2^k\}^*, 2^k > |x|^2 \)
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 $\Omega(k n)$ space.

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

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 $\Omega(kn)$ 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 $2^{\Omega(kn)}$

Proof: For each subset $T$ of $\Sigma$ of size $n/2$,
   define $x_T$ to be any concatenation of the strings 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 $2^{\Omega(kn)}$, for $2^k > n^2$. 
Randomized Algorithms Help!

The DE problem

Input: \( x \in \{0,1,\ldots,2^k\}^*, \ 2^k > |x|^2 \)

Output: The number of distinct elements appearing in \( x \)

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

Suppose: the elements are selected uniformly at random. What can we say about the minimal element?
If we have at our disposal a \textit{random permutation} \( h \) over \( \{0,1,\ldots,2^k\} \) what can we do?
Derandomization: \( h \) that is efficient and have short description.
Parting thoughts:
Myhill-Nerode Theorem – powerful characterization
Streaming – modern day incarnations of DFAs
Randomness – could be a useful resource of computation

Questions?