CS154

Lecture 6:
Streaming Algorithms and Communication Complexity
Streaming Algorithms
Streaming Algorithms
L = \{x \mid x \text{ has more 1’s than 0’s}\}

Initialize $C := 0$ and $B := 0$
Read the next bit $x$ from the stream
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

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 (to store $B$ and $C$)
Streaming Algorithms

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. Could also make multiple passes over the data, could be randomized

Can recognize non-regular languages
Theorem: Suppose a language $L$ can be recognized by a DFA with $\leq 2^p$ states. Then $L$ is computable by a streaming algorithm $A$ using $\leq p$ bits of space.

Proof Idea: Algorithm $A$ stores the DFA’s current state in memory, beginning with the start state. Alg. $A$ makes decisions based on DFA transitions. When the string ends, $A$ outputs $accept$ if the DFA state is accepting, $reject$ otherwise.
DFAs and Streaming

For any $L \subseteq \Sigma^*$ define $L_n = L \cap \Sigma^n$

**Theorem:** Suppose $L$ is computable by a streaming algorithm $A$ using $f(n)$ bits of space, on all strings of length $n$
Then for all $n$, $L_n$ is recognized by a DFA with $\leq 2^{f(n)}$ states.

**Proof Idea:** The new DFA will have a state for each of the $2^{f(n)}$ possible configurations of $A$’s memory. When $A$ sees a symbol, its memory will update; the transition function of the DFA can simulate that.
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 \((\log_2 n)-1\) 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 \(L_n \subseteq \{0,1\}^n \cap L\)

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

\(\Rightarrow\) Every streaming algorithm for L requires at least \((\log n)-1\) bits of memory
$L = \{x \mid x \text{ has more 1's than 0's}\}$

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

Suppose we partition all strings into their equivalence classes under $\equiv_{\binom{L}{n}}$

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

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

**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 \( S_n \), \( k > j \)

**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 \( yz \not\in L_n \)

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

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

**Proof (Slide 2):**

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

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

Then, from the Myhill-Nerode Theorem:

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

\[\Rightarrow\] 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!
Number of Distinct Elements

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

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
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!

See the lecture notes for more details.
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: Each bit sent is a function of the party’s input and all the bits communicated so far in the conversation.

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 in any protocol
for computing $f(x, y)$ over all $n$-bit $x, y$
Example. Let $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ be arbitrary

There is always a “trivial” protocol:

- Alice sends bits of $x$ in odd rounds
- Bob sends bits of $y$ in even rounds
- After $2n$ rounds, they both know each other’s input!

The communication complexity of every $f$ is at most $2n$
Example. \( \text{PARITY}(x, y) = \sum_i x_i + \sum_i y_i \mod 2. \)

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.

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

What’s a good protocol for computing MAJORITY?

Alice sends $b = \text{number of 1s in } x$

Bob computes $c = \text{number of 1s in } y$,

sends 1 iff $b + c$ is greater than $(|x| + |y|)/2 = n$

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

What’s a good protocol for computing EQUALS?

?????

*Communication complexity of 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:

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

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

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

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

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

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