CS154

Streaming Algorithms and Communication Complexity
Streaming Algorithms
Streaming Algorithms

42

$q$
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,

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

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

On all strings of length \( n \), the algorithm uses \( 1 + \log_2 (n+1) \) bits of space (\textit{to store } \( B \) \& \( C \))
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 $M$ 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: Use the transition function of $M$ to 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

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

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 $\leq 2^{f(n)}$ states such that $L'_n = L(M)_n$

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
Final states of $M =$ Memory configurations in which $A$ would accept, if the string ends
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,
\( \text{accept if } B=1 \text{ and } C > 0, \text{ else reject} \)

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 \textit{less than} \((\log_2 n)\) space?

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

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, let \( L_n = \{ x \in L : |x| = n \} \)

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'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 \(\equiv_{L_n}\)

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}\}

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 \( S_n \), 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 \( yz \not\in L_n \)

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 $(\log_2 n) - 1$ 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 $\equiv_{L_n}$
By 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: \( x \in \{0,1,\ldots,2^k\}^\ast, \; 2^k > |x|^2 \)

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

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

Theorem: Every streaming algorithm for DE requires \( \Omega(kn) \) 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 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: 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 $b_r = A(x, b_1 \ldots b_{r-1})$

else Bob sends $b_r = B(y, b_1 \ldots b_{r-1})$

Output $b_{r-1}$. Number of rounds $= r - 1$
Def. The *cost of a protocol P for f on n-bit strings is* \[
\max_{x,y \in \{0,1\}^n} \text{[number of rounds in P to compute } f(x, y)]
\]

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

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 $N_x = \text{number of 1s in } x$
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 $\text{EQUALS}$?

$\text{Communication complexity of } \text{EQUALS} \text{ is at most } 2n$