6.045

Lecture 7:
Streaming Algorithms and Communication Complexity

6.045

Announcements:

- Pset 2 is due tonight, 11:59pm
- Pest 3 is out!
  Due next Wednesday

L is regular

if and only if

(∃ DFA M)(∀ strings x)[M acc. x ⇔ x ∈ L]

"M gives the correct output on all strings"

L is NOT regular

if and only if

(∀ DFA M)(∃ string x_M)[M acc. x_M ⇔ x ∉ L]

"M gives the wrong output on x_M"

So the problem of proving L is NOT regular can be viewed as a problem about designing "bad inputs"

Distinguishing set for L

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 σ:

<pseudocode using σ and vars>

When stream stops (end of string):

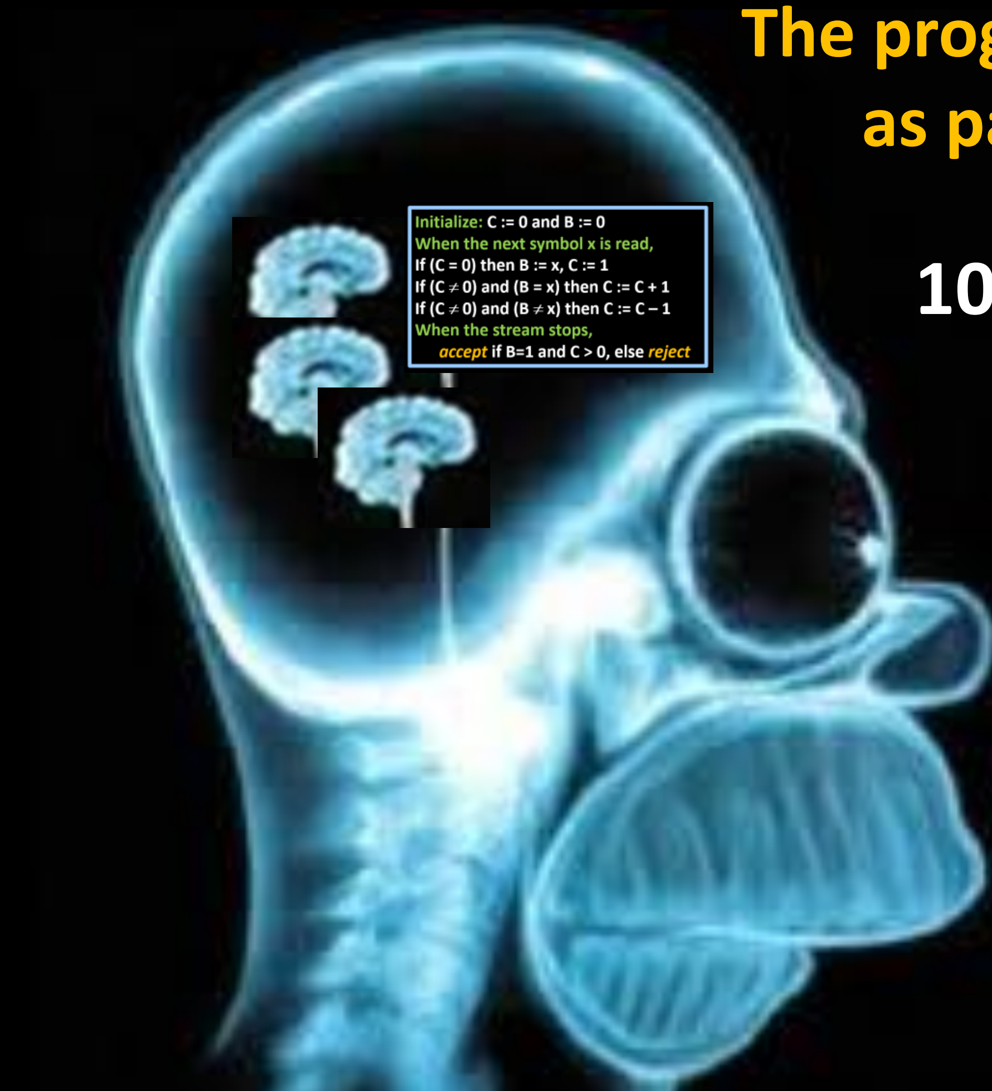
<*accept/reject* condition on vars>

(or: <pseudocode for output>)

Algorithm A computes $L \subseteq \Sigma^*$ if

A accepts the strings in L, rejects strings not in L

How to think of memory usage



The program is *not considered* as part of the memory

1010101111101011111110101

Space Usage of A:
$S(n)$ = maximum # of bits used to store vars in A, over all inputs of length *up to* n

DFAs and Streaming

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

Theorem: Let L' be computable by streaming algorithm A with space usage $\leq S(n)$. Then for all n, there is a DFA M with $< 2^{S(n)+1}$ states such that $L'_n = L(M)_n$

For all streaming algorithms A using *S*(n) space, and all n, there's a DFA M of $< 2^{S(n)+1}$ states such that A and M *agree* on all strings of length up to n.

Note: L'_n is always regular!
(It's a finite set!)

DFAs and Streaming

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

Theorem: Let L' be computable by streaming algorithm A with space usage $\leq S(n)$. Then for all n, there is a DFA M with $< 2^{S(n)+1}$ states such that $L'_n = L(M)_n$

Proof Idea: States of M = The set of (at most) $2^{S(n)+1} - 1$ memory configurations of A, over strings of length up to n

(Why $2^{S(n)+1} - 1$?)

Start state of M = Initialized memory of A

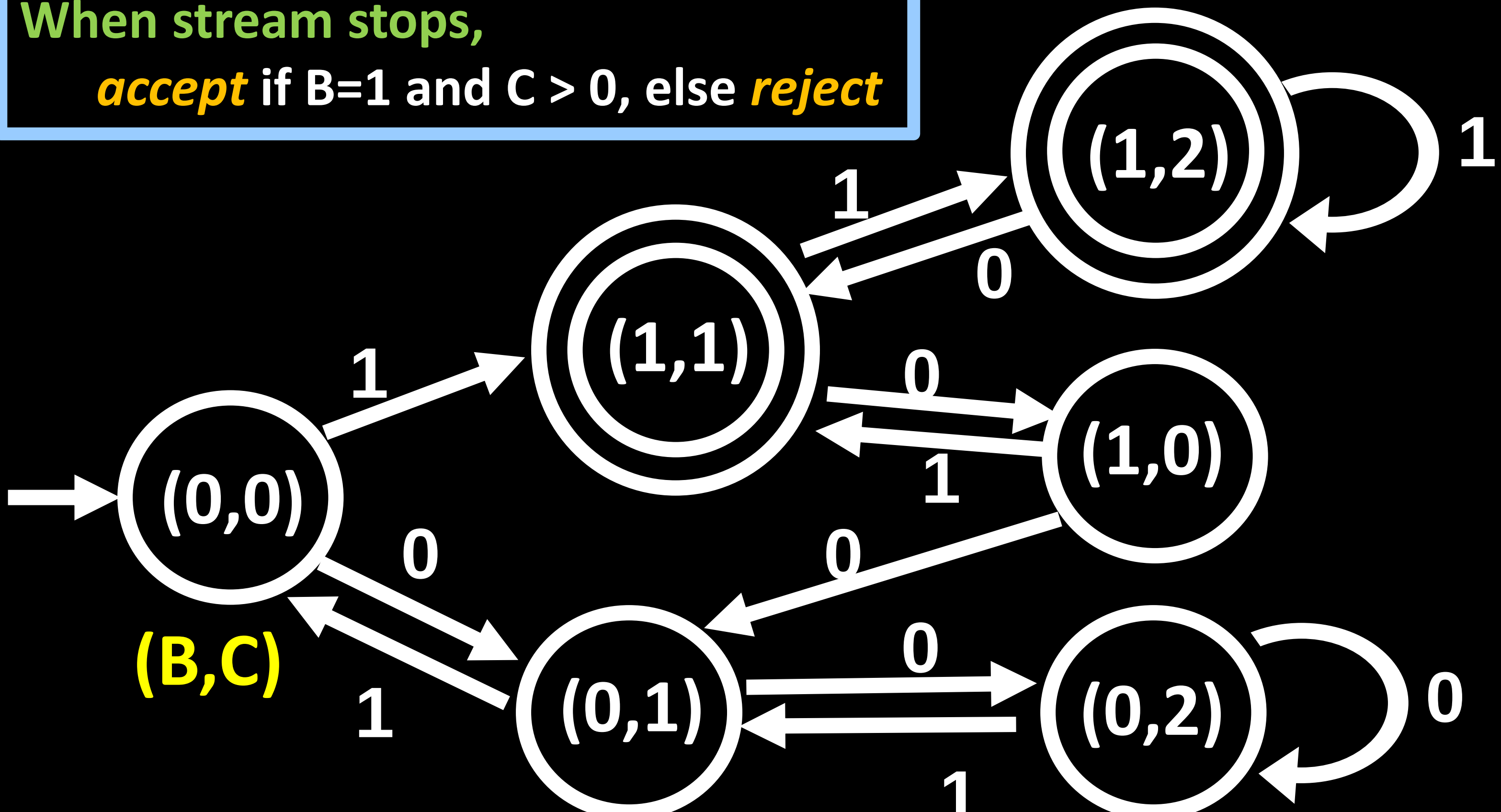
Transition function = Mimic how A updates its memory

Final states of M = Subset of memory configurations in which A would accept, if the string ended there

Example: L = {x | x has more 1's than 0's}

Initialize: C := 0 and B := 0
When next symbol seen is σ,
If (C = 0) then B := σ, C := 1
If (C $\neq$ 0) and (B = σ) then C := C + 1
If (C $\neq$ 0) and (B $\neq$ σ) then C := C – 1
When stream stops,
accept if B=1 and C > 0, else *reject*

Example: 6-state DFA that agrees with L on all strings of length ≤ 3
(We only let C go up to 2)



Distinguishing set for L

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

In fact, Myhill-Nerode shows that the size of a distinguishing set for L is a *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 similar ideas to prove lower bounds on streaming algorithms!

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

A **streaming distinguisher for** L_n is a subset D_n of Σ^*:
for all distinct $x, y \in D_n$, there is a z in Σ^* such that
$|xz| \leq n$, $|yz| \leq n$, and *exactly one* of xz, yz is in L.

Streaming Theorem: Suppose for all n, there is a streaming distinguisher D_n for L_n with $|D_n| \geq 2^{S(n)}$.
Then all streaming algs for L must use at least $S(n)$ space!

Idea: Use the set D_n to show that every streaming algorithm for L must enter at least $2^{S(n)}$ different memory states, over all inputs of length at most n.

But if there are at least $2^{S(n)}$ distinct memory states,
Then the alg must be using at least $S(n)$ bits of space!

$L = \{ 0^k 1^k \mid k \geq 0 \}$

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

Theorem: For all n, every streaming algorithm computing L must to use at least $\lfloor \log_2 n \rfloor$ bits of space.

Idea: Show there is a streaming distinguisher D_n for $L_n = \{ 0^k 1^k \mid 0 \leq k \leq n \}$ with $|D_n| = n/2+1$.
By the Streaming Theorem, it follows that all streaming algs for L need $\geq \log_2 (n/2+1)$ space!

$L = \{ 0^k 1^k \mid k \geq 0 \}$

Theorem: For all (even) n, every streaming algorithm computing L needs at least $\lfloor \log_2 n \rfloor$ bits of space.

Proof: For even n, let $D_n = \{0^i \mid i = 0, \ldots, n/2\}$

Claim: For all n, D_n is a *streaming distinguisher* for L_n

Let $x=0^a$ and $y=0^b$ be distinct strings in D_n. Set $z = 1^b$.
Then $yz \in L$, $xz \notin L$, and $|xz| \leq n$, $|yz| \leq n$. **QED**

Since $|D_n| = n/2+1$, Streaming Thm says: every streaming algorithm for L needs $\geq \log_2 (n/2+1)$ space.

Note $\log_2 (n/2+1) > \log_2 (n/2) = \log_2 (n) - 1$

"heavy hitters"

Finding Frequent Items

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?) $<k$

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

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 σ 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 σ occurring $> n/k$ times in x

2nd pass: Count occurrences of all σ' appearing in T to determine those occurring $> n/k$ times

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

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

When the next symbol σ 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 σ' appearing in T to determine those occurring > n/k times

Claim: At end, if σ is not in T then σ occurs $\leq$ n/k times

Idea: Have k-1 containers, n colored balls, and a trash can.

For each ball colored *σ*: either add it to a container, or *throw it in the trash along with k-1 other balls*, one from each container.

If there were m balls colored *σ*, and no balls of color *σ* are in containers at the end, there must be $k \cdot m \leq n$ balls in the trash!

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

When the next symbol σ 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 $(\sigma',m') \in T$, ← **When this happens**

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

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

Decrement k-1 counters

2nd pass: Count occurrences of all σ' 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 DE(x)

Ex: x = 12312 has DE(x) = 3 ($\Sigma=\{1,2,3\}$)

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

Theorem: Every streaming algorithm for DE requires Ω(k n) space!

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

Say $x, y \in \Sigma$ are *length-n DE distinguishable* if $(\exists z \in \Sigma^*)[DE(xz) \neq DE(yz)]$ & $|xz| \leq n$, $|yz| \leq n]$

Lemma: Let $S \subseteq \Sigma^*$ be such that every pair of strings in S is length-n DE distinguishable. Then, streaming algs for DE need $\geq \log_2 |S|$ bits of space (on inputs of length $\leq n$)

Proof Sketch: Let algorithm A use $< \log_2 |S|$ space. We show A cannot compute DE on all inputs of length $\leq n$.
By the 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 $DE(xz) \neq DE(yz)$, so A does not compute DE.

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

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

Claim: For all n, there is a such a set S with $|S| \geq 2^{k\ n/4}$

Proof: For each subset T of Σ 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

$\binom{2^k}{n/2} \geq 2^{k\ n/2}/(n/2)^{n/2} \geq 2^{k\ n/4}$, for $n < 2^{k/2}$

Theorem: Every streaming algorithm for DE requires Ω(k n) space

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

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

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

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

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

Since $\left(\frac{n}{2}\right)^{\frac{n}{2}} < \left(\frac{2^{\frac{k}{2}}}{2}\right)^{\frac{n}{2}} < 2^{\frac{kn}{4}}$, we have $\binom{2^k}{\frac{n}{2}} \geq \frac{2^{\frac{kn}{2}}}{\left(\frac{n}{2}\right)^{\frac{n}{2}}} > 2^{\frac{kn}{4}}$

Theorem: Every streaming algorithm for *approximating the number of* DE to within +- 20% error also requires Ω(k n) space!

See Lecture Notes.

Randomized Algorithms Help!

Distinct Elements (DE)
Input: $x \in \{1,\ldots,2^k\}^*$, $n = |x| < 2^{k/2}$
Output: The number of different 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 $h: \{1,\ldots,2^k\} \rightarrow [0, 1]$ 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 = minimum of $DE(x)$ random numbers in $[0,1]$

Claim: Let $x \in \{1,\ldots,2^k\}^*$

With probability > 0.8, $DE(x)/5 \leq 1/m \leq 10 \cdot DE(x)$.

Can boost accuracy using O(1) more hash functions!

(See the Lecture Notes!)

Randomized Algorithm for DE

Idea: **Let $h: \{1,\dots,2^k\} \rightarrow [0, 1]$ be a *random* function.**

(For all $i \in \{1,\dots,2^k\}$, pick $j \in \{1,\dots,n^2\}$ 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 the function 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.