Data Streams — A Tutorial

Nicole Schweikardt

Goethe-Universität Frankfurt am Main

DEIS'10: GI-Dagstuhl Seminar on Data Exchange, Integration, and Streams

Schloss Dagstuhl, November 8, 2010

Data Streams

Situation:

- massive amounts of data
- generated automatically
- continuous, rapid updates

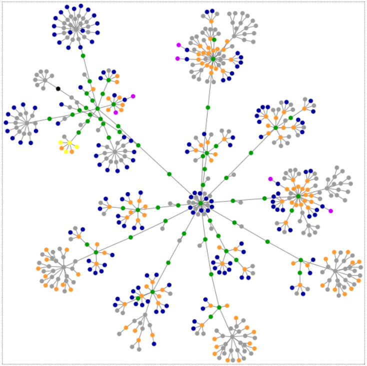
Examples:

- meteorological data (sensor networks)
- astronomical data
- network monitoring
- banking and credit transactions

Challenges:

- cannot wait with processing until "all" the data has arrived
  $\rightsquigarrow$ process data "on-the-fly"
- cannot afford to store all the data $\rightsquigarrow$ store a "sketch"
- data may arrive so rapidly that you cannot even afford to look at each incoming data item $\rightsquigarrow$ "sampling"

Example: Network Monitoring



Let A be a node in the world wide web.
As input, A receives a stream of "packets"

$$p_1, p_2, p_3, p_4, \ldots, p_m.$$

Each packet p_i contains information on

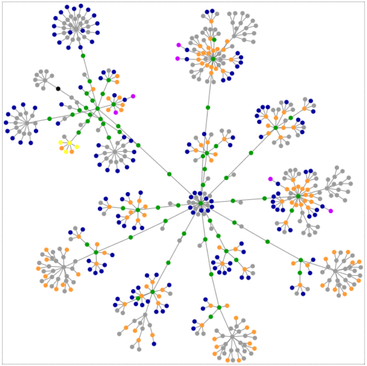
- the sender's IP address,
- the destination's IP address,
- the data that is transmitted

Question: How many distinct IP addresses have sent at least one packet through node A? — I.e., what is the 0-th frequency moment F_0 of the input stream?

Problem: A does not want to store the entire stream $p_1, p_2, p_3, \ldots, p_m$.

Solution:
A suitable randomised algorithm that computes a good approximate answer:

Example: Network Monitoring



Let A be a node in the world wide web.
As input, A receives a stream of "packets"

$$p_1, p_2, p_3, p_4, \ldots, p_m.$$

Each packet p_i contains information on

- the sender's IP address,
- the destination's IP address,
- the data that is transmitted

Question: How many distinct IP addresses have sent at least one packet through node A? — I.e., what is the 0-th frequency moment F_0 of the input stream?

Problem: A does not want to store the entire stream $p_1, p_2, p_3, \ldots, p_m$.

Solution:
A suitable randomised algorithm that computes a good approximate answer:

Tight bound for approximating F_0

COMPUTING F_0

Input: A sequence $p_1, p_2, p_3, \ldots, p_m$ of elements in $\{1, \ldots, n\}$.

Task: Compute the number F_0 of *distinct* elements in the input.

Theorem:

(a) *Upper Bound:* *(Flajolet, Martin, FOCS'83)*

For every $c > 2$ there is a randomized one-pass algorithm that uses $O(\log n)$ bits of memory and computes a number Y such that
Prob $\left(\frac{Y}{F_0} \leqslant \frac{1}{c} \text{ or } \frac{Y}{F_0} \geqslant c\right) \leqslant 2/c$.

(b) *Lower Bound:* *(Alon, Matias, Szegedy, STOC'96)*

Any randomized one-pass algorithm computing a number Y such that *Prob* $\left(\frac{Y}{F_0} \leqslant 0.9 \text{ or } \frac{Y}{F_0} \geqslant 1.1\right) \leqslant 0.25$ *uses* $\Omega(\log n)$ *bits of memory.*

Remark: improved bounds: Bar-Yossef, Jayram, Kumar, Sivakumar (RANDOM'00) and Kane, Nelson, Woodruff (PODS'10).

Main issues concerning data streams:

How to design algorithms & how to prove lower bounds

Tight bound for approximating F_0

COMPUTING F_0

Input: A sequence $p_1, p_2, p_3, \ldots, p_m$ of elements in $\{1, \ldots, n\}$.

Task: Compute the number F_0 of *distinct* elements in the input.

Theorem:

(a) *Upper Bound:* *(Flajolet, Martin, FOCS'83)*

For every $c > 2$ there is a randomized one-pass algorithm that uses $O(\log n)$ bits of memory and computes a number Y such that $Prob\left(\frac{Y}{F_0} \leqslant \frac{1}{c} \text{ or } \frac{Y}{F_0} \geqslant c\right) \leqslant 2/c$.

(b) *Lower Bound:* *(Alon, Matias, Szegedy, STOC'96)*

Any randomized one-pass algorithm computing a number Y such that $Prob\left(\frac{Y}{F_0} \leqslant 0.9 \text{ or } \frac{Y}{F_0} \geqslant 1.1\right) \leqslant 0.25$ *uses $\Omega(\log n)$ bits of memory.*

Remark: improved bounds: Bar-Yossef, Jayram, Kumar, Sivakumar (RANDOM'00) and Kane, Nelson, Woodruff (PODS'10).

Main issues concerning data streams:

How to design algorithms & how to prove lower bounds

Tight bound for approximating F_0

COMPUTING F_0

Input: A sequence $p_1, p_2, p_3, \ldots, p_m$ of elements in $\{1, \ldots, n\}$.

Task: Compute the number F_0 of *distinct* elements in the input.

Theorem:

(a) *Upper Bound:* *(Flajolet, Martin, FOCS'83)*

For every $c > 2$ there is a randomized one-pass algorithm that uses $O(\log n)$ bits of memory and computes a number Y such that $\mathit{Prob}\left(\frac{Y}{F_0} \leqslant \frac{1}{c} \text{ or } \frac{Y}{F_0} \geqslant c\right) \leqslant 2/c$.

(b) *Lower Bound:* *(Alon, Matias, Szegedy, STOC'96)*

Any randomized one-pass algorithm computing a number Y such that $\mathit{Prob}\left(\frac{Y}{F_0} \leqslant 0.9 \text{ or } \frac{Y}{F_0} \geqslant 1.1\right) \leqslant 0.25$ uses $\Omega(\log n)$ bits of memory.

Remark: improved bounds: Bar-Yossef, Jayram, Kumar, Sivakumar (RANDOM'00) and Kane, Nelson, Woodruff (PODS'10).

Main issues concerning data streams:

How to design algorithms & how to prove lower bounds

Overview

One pass over a single stream

Several passes over a single stream

Several passes over several streams in parallel

Read/write streams

Future tasks

Overview

One pass over a single stream

Several passes over a single stream

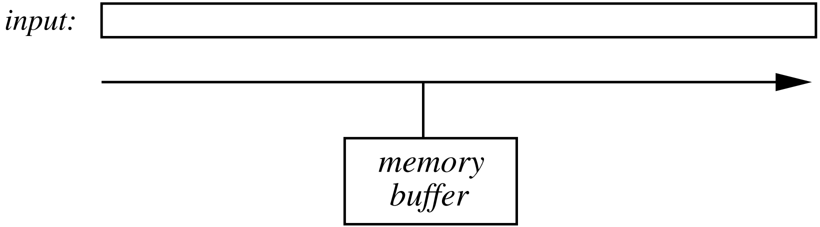
Several passes over several streams in parallel

Read/write streams

Future tasks

One pass over a single stream

Scenario:



Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
	✓								

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
	✓			✓					

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
$\checkmark$	$\checkmark$			$\checkmark$					

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓		✓					

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓					

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n{-}1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓			✓		

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓		

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Clever Solution: Store running sum $O(\log n)$ bits suffice

$$s := x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1}$$

$$\text{Missing number} = \frac{n \cdot (n+1)}{2} - s$$

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Clever Solution: Store running sum $O(\log n)$ bits suffice

$$s := x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1}$$

$$\text{Missing number} = \frac{n \cdot (n+1)}{2} - s$$

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Clever Solution: Store running sum $O(\log n)$ bits suffice

$$s \;:=\; x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1}$$

$$\text{Missing number} \;=\; \frac{n \cdot (n+1)}{2} - s$$

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Clever Solution: Store running sum $O(\log n)$ bits suffice

$$s := x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1}$$

$$\text{Missing number} = \frac{n \cdot (n+1)}{2} - s$$

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n{-}1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n — requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Clever Solution: Store running sum — $O(\log n)$ bits suffice

$$s \;:=\; x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1}$$

$$\text{Missing number} \;=\; \frac{n \cdot (n{+}1)}{2} - s$$

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Clever Solution: Store running sum $O(\log n)$ bits suffice

$$s := x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1}$$

$$\text{Missing number} = \frac{n \cdot (n+1)}{2} - s$$

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n{-}1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n — requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Clever Solution: Store running sum — $O(\log n)$ bits suffice

$$s \;:=\; x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1}$$

$$\text{Missing number} \;=\; \frac{n \cdot (n{+}1)}{2} \;-\; s$$

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n — requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Clever Solution: Store running sum — $O(\log n)$ bits suffice

$$s \;:=\; x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1}$$

$$\text{Missing number} \;=\; \frac{n \cdot (n+1)}{2} - s$$

Lower Bound:

Missing Number Puzzle

MISSING NUMBER

Input: A stream $x_1, x_2, x_3, \ldots, x_{n-1}$ of $n-1$ distinct numbers from $\{1, \ldots, n\}$.

Question: Which number from $\{1, \ldots, n\}$ is missing?

Naive Solution: 2 5 1 3 4 8 6 $\cdots$ n requires n bits of storage

1	2	3	4	5	6	7	8	$\cdots$	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

Clever Solution: Store running sum $O(\log n)$ bits suffice

$$s \;:=\; x_1 + x_2 + x_3 + x_4 + \cdots + x_{n-1}$$

$$\text{Missing number} \;=\; \frac{n \cdot (n+1)}{2} \;-\; s$$

Lower Bound: at least $\log n$ bits are necessary

Exercise # 1

Find a data stream algorithm that uses at most poly($k \cdot \log n$) bits of memory and solves the following generalization of the "missing numbers puzzle":

k MISSING NUMBERS

Input: Two numbers n, k and a stream $x_1, x_2, x_3, \ldots, x_{n-k}$ of $n-k$ distinct numbers from $\{1, \ldots, n\}$

Task: Find the k missing numbers

The MULTISET-EQUALITY Problem (1/3)

MULTISET-EQUALITY *Total input length:* $N = O(m \cdot \log n)$ *bits*

Input: Two multisets $\{x_1, \ldots, x_m\}$ and $\{y_1, \ldots, y_m\}$ of numbers x_i, y_j in $\{1, \ldots, n\}$.

Question: Is $\{x_1, \ldots, x_m\} = \{y_1, \ldots, y_m\}$?

Observation:

Every deterministic solution requires $\Omega(N)$ bits of storage.

Proof:

- Use fact from Communication Complexity:

Communication Complexity

Yao's 2-Party Communication Model:

- 2 players: Alice & Bob
- both know a function $f : A \times B \to \{0,1\}$
- Alice only sees input $a \in A$, Bob only sees input $b \in B$
- they jointly want to compute $f(a,b)$
- Goal: exchange as few bits of communication as possible

Fact: Deciding if two m-element input sets

$$a = \{x_1, \ldots, x_m\} \subseteq \{1, \ldots, n\} \quad \text{und} \quad b = \{y_1, \ldots, y_m\} \subseteq \{1, \ldots, n\}$$

are equal, requires at least $\log\binom{n}{m}$ bits of communication.

Communication Complexity

Yao's 2-Party Communication Model:

- 2 players: Alice & Bob
- both know a function $f : A \times B \to \{0, 1\}$
- Alice only sees input $a \in A$, Bob only sees input $b \in B$
- they jointly want to compute $f(a, b)$
- Goal: exchange as few bits of communication as possible

Fact: Deciding if two m-element input sets

$$a = \{x_1, \ldots, x_m\} \subseteq \{1, \ldots, n\} \quad \text{und} \quad b = \{y_1, \ldots, y_m\} \subseteq \{1, \ldots, n\}$$

are equal, requires at least $\log\binom{n}{m}$ bits of communication.

The MULTISET-EQUALITY Problem (1/3)

MULTISET-EQUALITY *Total input length:* $N = O(m \cdot \log n)$ *bits*

Input: Two multisets $\{x_1, \ldots, x_m\}$ and $\{y_1, \ldots, y_m\}$ of numbers x_i, y_j in $\{1, \ldots, n\}$.

Question: Is $\{x_1, \ldots, x_m\} = \{y_1, \ldots, y_m\}$?

Observation:

Every deterministic solution requires $\Omega(N)$ bits of storage.

Proof:

- Use fact from Communication Complexity:

The MULTISET-EQUALITY Problem (1/3)

MULTISET-EQUALITY *Total input length: $N = O(m \cdot \log n)$ bits*

Input: Two multisets $\{x_1, \ldots, x_m\}$ and $\{y_1, \ldots, y_m\}$ of numbers x_i, y_j in $\{1, \ldots, n\}$.

Question: Is $\{x_1, \ldots, x_m\} = \{y_1, \ldots, y_m\}$?

Observation:

Every deterministic solution requires $\Omega(N)$ bits of storage.

Proof:

- Use fact from Communication Complexity:
  Deciding if two m-element subsets of $\{1, \ldots, n\}$ are equal requires at least $\log\binom{n}{m}$ bits of communication.
- If $n = m^2$, then $\log\binom{n}{m} \geqslant m \cdot \log m$ bits of communication are necessary, and the total length of the corresponding MULTISET-EQUALITY input is $N = \Theta(m \cdot \log m)$.

The MULTISET-EQUALITY Problem (1/3)

MULTISET-EQUALITY *Total input length:* $N = O(m \cdot \log n)$ *bits*

Input: Two multisets $\{x_1, \ldots, x_m\}$ and $\{y_1, \ldots, y_m\}$ of numbers x_i, y_j in $\{1, \ldots, n\}$.

Question: Is $\{x_1, \ldots, x_m\} = \{y_1, \ldots, y_m\}$?

Observation:

Every deterministic solution requires $\Omega(N)$ bits of storage.

Proof:

- Use fact from Communication Complexity:
  Deciding if two m-element subsets of $\{1, \ldots, n\}$ are equal requires at least $\log\binom{n}{m}$ bits of communication.
- If $n = m^2$, then $\log\binom{n}{m} \geqslant m \cdot \log m$ bits of communication are necessary, and the total length of the corresponding MULTISET-EQUALITY input is $N = \Theta(m \cdot \log m)$.

The MULTISET-EQUALITY Problem (2/3)

Proof (continued):

- Known: $N = \Theta(m \cdot \log m)$, and $\geqslant m \cdot \log m$ bits of communication are necessary for solving MULTISET-EQUALITY.
- A deterministic data stream algorithm solving MULTISET-EQUALITY with s bits of storage would lead to a communication protocol with s bits of communication.

- Thus: Lower bound on communication complexity $\leadsto$ lower bound on memory size of data stream algorithm

The Multiset-Equality Problem (2/3)

Proof (continued):

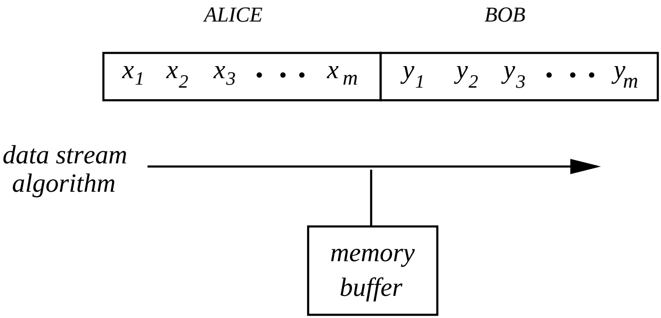
- Known: $N = \Theta(m \cdot \log m)$, and $\geqslant m \cdot \log m$ bits of communication are necessary for solving Multiset-Equality.
- A deterministic data stream algorithm solving Multiset-Equality with s bits of storage would lead to a communication protocol with s bits of communication.

- Thus: Lower bound on communication complexity $\rightsquigarrow$ lower bound on memory size of data stream algorithm

The MULTISET-EQUALITY Problem (2/3)

Proof (continued):

- Known: $N = \Theta(m \cdot \log m)$, and $\geqslant m \cdot \log m$ bits of communication are necessary for solving MULTISET-EQUALITY.
- A deterministic data stream algorithm solving MULTISET-EQUALITY with s bits of storage would lead to a communication protocol with s bits of communication.

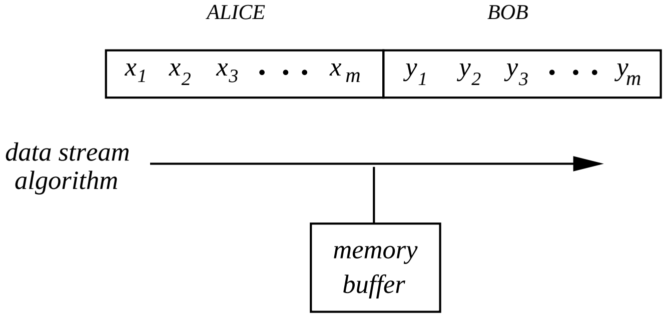


- Thus: Lower bound on communication complexity $\rightsquigarrow$ lower bound on memory size of data stream algorithm

The MULTISET-EQUALITY Problem (2/3)

Proof (continued):

- Known: $N = \Theta(m \cdot \log m)$, and $\geqslant m \cdot \log m$ bits of communication are necessary for solving MULTISET-EQUALITY.
- A deterministic data stream algorithm solving MULTISET-EQUALITY with s bits of storage would lead to a communication protocol with s bits of communication.



- Thus: Lower bound on communication complexity $\rightsquigarrow$ lower bound on memory size of data stream algorithm

The MULTISET-EQUALITY Problem (3/3)

Theorem: *(Grohe, Hernich, S., PODS'06)*

The MULTISET-EQUALITY *problem can be solved by a randomised algorithm using $O(\log N)$ bits of storage in the following sense:*

Given m, n, and a stream of numbers $a_1, \ldots, a_m,\ b_1, \ldots, b_m$ from $\{1, \ldots, n\}$, the algorithm

- *accepts with probability 1 if $\{a_1, \ldots, a_m\} = \{b_1, \ldots, b_m\}$*
- *rejects with probability $\geqslant 0.9$ if $\{a_1, \ldots, a_m\} \neq \{b_1, \ldots, b_m\}$.*

Basic idea: Use "Fingerprinting"-techniques:

- represent $\{a_1, \ldots, a_m\}$ by a polynomial $f(x) := \sum_{i=1}^{m} x^{a_i}$
- represent $\{b_1, \ldots, b_m\}$ by a polynomial $g(x) := \sum_{i=1}^{m} x^{b_i}$
- choose a random number r and check if $f(r) = g(r)$
- accept if $f(r) = g(r)$; reject otherwise.

If $\{a_1, \ldots, a_m\} = \{b_1, \ldots, b_m\}$, then $f(x) = g(x)$, and thus the algorithm always accepts. If $\{a_1, \ldots, a_m\} \neq \{b_1, \ldots, b_m\}$, then there are at most *degree*$(f-g)$ many distinct r with $f(r) = g(r)$, and thus the algorithm rejects with high probability.

The MULTISET-EQUALITY Problem (3/3)

Theorem: *(Grohe, Hernich, S., PODS'06)*

The MULTISET-EQUALITY *problem can be solved by a randomised algorithm using $O(\log N)$ bits of storage in the following sense:*

Given m, n, and a stream of numbers $a_1, \ldots, a_m,\ b_1, \ldots, b_m$ from $\{1, \ldots, n\}$, the algorithm

- *accepts with probability 1 if $\{a_1, \ldots, a_m\} = \{b_1, \ldots, b_m\}$*
- *rejects with probability $\geqslant 0.9$ if $\{a_1, \ldots, a_m\} \neq \{b_1, \ldots, b_m\}$.*

Basic idea: Use "Fingerprinting"-techniques:

- represent $\{a_1, \ldots, a_m\}$ by a polynomial $f(x) := \sum_{i=1}^{m} x^{a_i}$
- represent $\{b_1, \ldots, b_m\}$ by a polynomial $g(x) := \sum_{i=1}^{m} x^{b_i}$
- choose a random number r and check if $f(r) = g(r)$
- accept if $f(r) = g(r)$; reject otherwise.

If $\{a_1, \ldots, a_m\} = \{b_1, \ldots, b_m\}$, then $f(x) = g(x)$, and thus the algorithm always accepts. If $\{a_1, \ldots, a_m\} \neq \{b_1, \ldots, b_m\}$, then there are at most $degree(f-g)$ many distinct r with $f(r) = g(r)$, and thus the algorithm rejects with high probability.

The MULTISET-EQUALITY Problem (3/3)

Theorem: *(Grohe, Hernich, S., PODS'06)*

The MULTISET-EQUALITY *problem can be solved by a randomised algorithm using $O(\log N)$ bits of storage in the following sense:*

Given m, n, and a stream of numbers $a_1, \ldots, a_m,\ b_1, \ldots, b_m$ from $\{1, \ldots, n\}$, the algorithm

- *accepts with probability 1* *if* $\{a_1, \ldots, a_m\} = \{b_1, \ldots, b_m\}$
- *rejects with probability* $\geqslant 0.9$ *if* $\{a_1, \ldots, a_m\} \neq \{b_1, \ldots, b_m\}$.

Basic idea: Use "Fingerprinting"-techniques:

- represent $\{a_1, \ldots, a_m\}$ by a polynomial $f(x) := \sum_{i=1}^{m} x^{a_i}$
- represent $\{b_1, \ldots, b_m\}$ by a polynomial $g(x) := \sum_{i=1}^{m} x^{b_i}$
- choose a random number r and check if $f(r) = g(r)$
- accept if $f(r) = g(r)$; reject otherwise.

If $\{a_1, \ldots, a_m\} = \{b_1, \ldots, b_m\}$, then $f(x) = g(x)$, and thus the algorithm always accepts. If $\{a_1, \ldots, a_m\} \neq \{b_1, \ldots, b_m\}$, then there are at most *degree*$(f-g)$ many distinct r with $f(r) = g(r)$, and thus the algorithm rejects with high probability.

The MULTISET-EQUALITY Problem (3/3)

Theorem: *(Grohe, Hernich, S., PODS'06)*

The MULTISET-EQUALITY *problem can be solved by a randomised algorithm using* $O(\log N)$ *bits of storage in the following sense:*

Given m, n*, and a stream of numbers* $a_1, \ldots, a_m,\ b_1, \ldots, b_m$ *from* $\{1, \ldots, n\}$*, the algorithm*

- *accepts with probability 1* if $\{a_1, \ldots, a_m\} = \{b_1, \ldots, b_m\}$
- *rejects with probability* $\geqslant 0.9$ *if* $\{a_1, \ldots, a_m\} \neq \{b_1, \ldots, b_m\}$.

Basic idea: Use "Fingerprinting"-techniques:

- represent $\{a_1, \ldots, a_m\}$ by a polynomial $f(x) := \sum_{i=1}^{m} x^{a_i}$
- represent $\{b_1, \ldots, b_m\}$ by a polynomial $g(x) := \sum_{i=1}^{m} x^{b_i}$
- choose a random number r and check if $f(r) = g(r)$
- accept if $f(r) = g(r)$; reject otherwise.

If $\{a_1, \ldots, a_m\} = \{b_1, \ldots, b_m\}$, then $f(x) = g(x)$, and thus the algorithm always accepts. If $\{a_1, \ldots, a_m\} \neq \{b_1, \ldots, b_m\}$, then there are at most *degree*$(f-g)$ many distinct r with $f(r) = g(r)$, and thus the algorithm rejects with high probability.

The MULTISET-EQUALITY Problem (3/3)

Theorem: *(Grohe, Hernich, S., PODS'06)*

The MULTISET-EQUALITY *problem can be solved by a randomised algorithm using $O(\log N)$ bits of storage in the following sense:*

Given m, n, and a stream of numbers $a_1, \ldots, a_m$, $b_1, \ldots, b_m$ from $\{1, \ldots, n\}$, the algorithm

- *accepts with probability 1 if $\{a_1, \ldots, a_m\} = \{b_1, \ldots, b_m\}$*
- *rejects with probability $\geqslant 0.9$ if $\{a_1, \ldots, a_m\} \neq \{b_1, \ldots, b_m\}$.*

Basic idea: Use "Fingerprinting"-techniques:

- represent $\{a_1, \ldots, a_m\}$ by a polynomial $f(x) := \sum_{i=1}^{m} x^{a_i}$
- represent $\{b_1, \ldots, b_m\}$ by a polynomial $g(x) := \sum_{i=1}^{m} x^{b_i}$
- choose a random number r and check if $f(r) = g(r)$
- accept if $f(r) = g(r)$; reject otherwise.

If $\{a_1, \ldots, a_m\} = \{b_1, \ldots, b_m\}$, then $f(x) = g(x)$, and thus the algorithm always accepts. If $\{a_1, \ldots, a_m\} \neq \{b_1, \ldots, b_m\}$, then there are at most *degree*$(f-g)$ many distinct r with $f(r) = g(r)$, and thus the algorithm rejects with high probability.

Exercise # 2

Work out the details of the described algorithm and its analysis.

Overview

One pass over a single stream

Several passes over a single stream

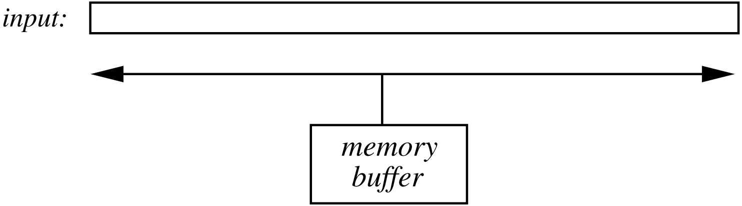
Several passes over several streams in parallel

Read/write streams

Future tasks

Several passes over a single stream

Scenario:



Parameters:

p : number of passes

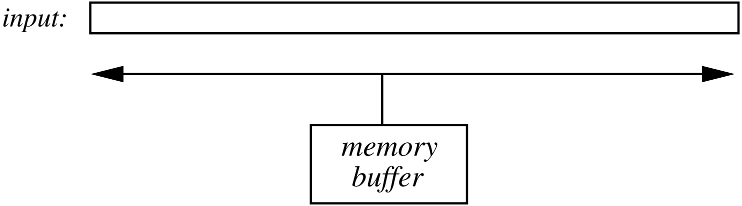
s : size of memory buffer (number of bits)

We call such computations (p, s)-bounded computations.

If necessary, an output stream can be generated during a computation.

Several passes over a single stream

Scenario:



Parameters:

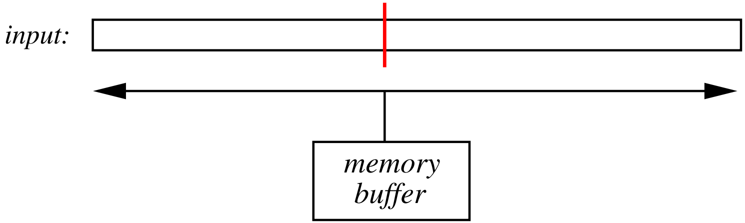
p : number of passes

s : size of memory buffer (number of bits)

We call such computations (p, s)-bounded computations.

If necessary, an output stream can be generated during a computation.

An easy observation



Fact:

During a (p, s)-bounded computation, only $(p \cdot s)$ bits can be communicated between the first and the second half of the input.

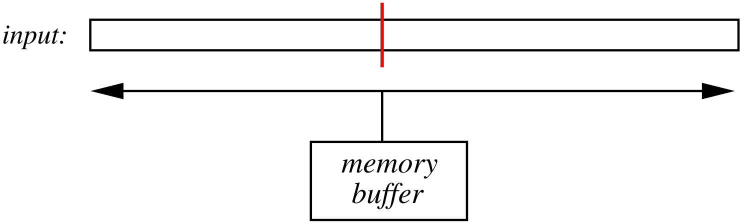
Consequence:

Lower bounds on communication complexity lead to lower bounds for (p, s)-bounded computations

... even if backward passes are allowed

... even if writing on the "input tape" is allowed.

An easy observation



Fact:

During a (p, s)-bounded computation, only $(p \cdot s)$ bits can be communicated between the first and the second half of the input.

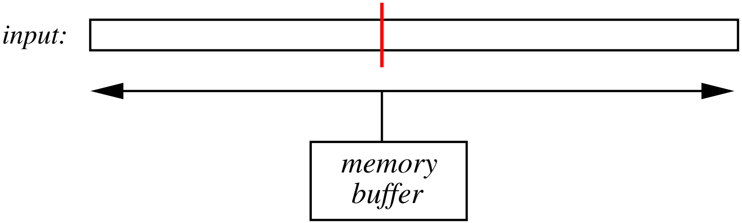
Consequence:

Lower bounds on communication complexity lead to lower bounds for (p, s)-bounded computations

. . . even if backward passes are allowed

. . . even if writing on the "input tape" is allowed.

An easy observation



Fact:

During a (p, s)-bounded computation, only $(p \cdot s)$ bits can be communicated between the first and the second half of the input.

Consequence:

Lower bounds on communication complexity lead to lower bounds for (p, s)-bounded computations

... even if backward passes are allowed

... even if writing on the "input tape" is allowed.

A lower bound for connectedness of a graph

CONNECTEDNESS *Parameters: m edges on $\leqslant n$ nodes*

Input: A list of edges $e_1, \ldots, e_m$ on node set $V \subseteq \{1, .., n\}$.

Question: Is the input graph connected?

Theorem: *(Henzinger, Raghavan, Rajagopalan, 1998)*

Solving CONNECTEDNESS *with p passes requires $\Omega(n/p)$ bits of memory.*

Proof:

By a reduction using the set disjointness problem.

SET DISJOINTNESS PROBLEM

Input: Two sets $A, B \subseteq \{1, .., n\}$

Question: Is $A \cap B = \emptyset$?

Known communication complexity of the set disjointness problem:
n bits of communication are necessary (and sufficient).

A lower bound for connectedness of a graph

CONNECTEDNESS *Parameters: m edges on $\leqslant n$ nodes*

Input: A list of edges $e_1, \ldots, e_m$ on node set $V \subseteq \{1, \ldots, n\}$.

Question: Is the input graph connected?

Theorem: *(Henzinger, Raghavan, Rajagopalan, 1998)*

Solving CONNECTEDNESS *with p passes requires* $\Omega(n/p)$ *bits of memory.*

Proof:

By a reduction using the set disjointness problem.

SET DISJOINTNESS PROBLEM

Input: Two sets $A, B \subseteq \{1, \ldots, n\}$

Question: Is $A \cap B = \emptyset$?

Known communication complexity of the set disjointness problem:
n bits of communication are necessary (and sufficient).

Exercise # 3

Work out the details of the proof:

(a) prove that n bits of communication are necessary for solving the set disjointness problem in Yao's 2-party communication model, and

(b) use this to show that solving graph connectedness with p passes requires $\Omega(n/p)$ bits of memory.

A lower bound for sorting

SORTING *Input length* $N = O(m \cdot \log n)$ *bits*

Input: A sequence of numbers $x_1, \ldots, x_m \in \{1, \ldots, n\}$ (for arbitrary m, n).

Output: $x_1, \ldots, x_m$ sorted in ascending order.

Theorem: *(Grohe, Koch, S., ICALP'05)*

SORTING *can be solved by a* (p, s)*-bounded computation* $\iff$ $(p \cdot s) \in \Omega(N)$

Proof:

- upper bound: easy.
- lower bound: by a reduction using the set disjointness problem.

A hierarchy on the number of passes

Allowing a single extra scan may be more powerful than significantly increasing the internal memory space:

Theorem: *(Hernich, S., Theor. Comput. Sci. 2008)*

For every logspace-computable function p with $p(N) \in o\left(\frac{N}{\log^2 N}\right)$, there exists a decision problem that

- *can be solved by a $(p+1, s)$-bounded computation, but*
- *that cannot be solved by any (p, S)-bounded computation,*

for $s(N) = O(\log N)$ and $S(N) = o\left(\frac{N}{p(N) \cdot \log N}\right)$.

Remark: An analogous result also holds for randomised computations.

Proof idea:

Use a result by Nisan and Wigderson (1993) on the k-round communication complexity of a particular "pointer jumping" problem.

A hierarchy on the number of passes

Allowing a single extra scan may be more powerful than significantly increasing the internal memory space:

Theorem: *(Hernich, S., Theor. Comput. Sci. 2008)*

For every logspace-computable function p with $p(N) \in o\left(\frac{N}{\log^2 N}\right)$, there exists a decision problem that

- *can be solved by a $(p+1, s)$-bounded computation, but*
- *that cannot be solved by any (p, S)-bounded computation,*

for $s(N) = O(\log N)$ and $S(N) = o\left(\frac{N}{p(N) \cdot \log N}\right)$.

Remark: An analogous result also holds for randomised computations.

Proof idea:

Use a result by Nisan and Wigderson (1993) on the *k*-round communication complexity of a particular "pointer jumping" problem.

A hierarchy on the number of passes

Allowing a single extra scan may be more powerful than significantly increasing the internal memory space:

Theorem: *(Hernich, S., Theor. Comput. Sci. 2008)*

For every logspace-computable function p with $p(N) \in o\left(\frac{N}{\log^2 N}\right)$*, there exists a decision problem that*

- *can be solved by a* $(p+1, s)$*-bounded computation, but*
- *that cannot be solved by any* (p, S)*-bounded computation,*

for $s(N) = O(\log N)$ *and* $S(N) = o\left(\frac{N}{p(N) \cdot \log N}\right)$.

Remark: An analogous result also holds for randomised computations.

Proof idea:

Use a result by Nisan and Wigderson (1993) on the k-round communication complexity of a particular “pointer jumping” problem.

A lower bound for finding a longest increasing subsequence

LONGEST-INCREASING-SUBSEQUENCE

Input: a sequence of numbers $x_1, \ldots, x_m \in \{1, \ldots, n\}$ (for arbitrary m, n)

Output: an increasing subsequence $x_{i_1}, \ldots, x_{i_k}$ of maximum length (denoted k)

Theorem: *(Guha, McGregor, ICALP'08)*

Any randomized p-pass algorithm solving LONGEST-INCREASING-SUBSEQUENCE *with p passes (and probability 0.9) requires* $\Omega\left(k^{1+\frac{1}{2^p-1}}\right)$ *bits of memory.*

Proof:

- not by using communication complexity
- introduce a new method of pass elimination (somewhat related to "round elimination" methods in communication complexity, but taylored towards stream processing).

Remark:

A matching upper bound was proved by Liben-Nowell, Vee, Zhu, COCOON'05.

A lower bound for finding a longest increasing subsequence

LONGEST-INCREASING-SUBSEQUENCE

Input: a sequence of numbers $x_1, \ldots, x_m \in \{1, \ldots, n\}$ (for arbitrary m, n)

Output: an increasing subsequence $x_{i_1}, \ldots, x_{i_k}$ of maximum length (denoted k)

Theorem: *(Guha, McGregor, ICALP'08)*

Any randomized p-pass algorithm solving LONGEST-INCREASING-SUBSEQUENCE *with p passes (and probability 0.9) requires* $\Omega\left(k^{1+\frac{1}{2^p-1}}\right)$ *bits of memory.*

Proof:

- not by using communication complexity
- introduce a new method of pass elimination (somewhat related to "round elimination" methods in communication complexity, but taylored towards stream processing).

Remark:

A matching upper bound was proved by Liben-Nowell, Vee, Zhu, COCOON'05.

Overview

One pass over a single stream

Several passes over a single stream

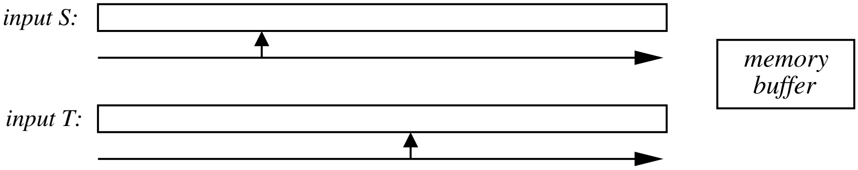
Several passes over several streams in parallel

Read/write streams

Future tasks

Several passes over several streams in parallel

Basic scenario:

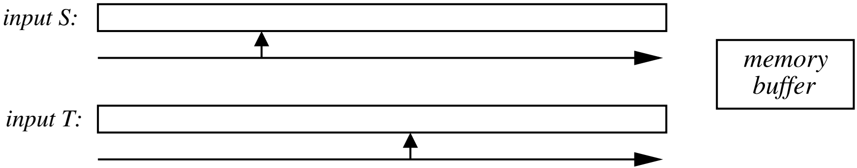


Parameters:

- 2 input streams: $S = s_1, s_2, \ldots, s_n$ and $T = t_1, t_2, \ldots, t_n$.
- one pass over each input; heads may proceed asynchronously
- advancement of heads and new content of memory depends on the current content of memory and the symbols seen at both heads
- for simplicity: advancement of only one head at a time
- s : size of memory buffer (number of bits)
- m : number of possible memory configurations, i.e., $\log m = s$

Several passes over several streams in parallel

Basic scenario:

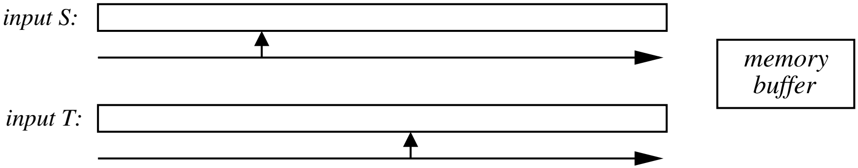


Parameters:

- 2 input streams: $S = s_1, s_2, \ldots, s_n$ and $T = t_1, t_2, \ldots, t_n$.
- one pass over each input; heads may proceed asynchronously
- advancement of heads and new content of memory depends on the current content of memory and the symbols seen at both heads
- for simplicity: advancement of only one head at a time
- s : size of memory buffer (number of bits)
- m : number of possible memory configurations, i.e., $\log m = s$

Several passes over several streams in parallel

Basic scenario:



Parameters:

- 2 input streams: $S = s_1, s_2, \ldots, s_n$ and $T = t_1, t_2, \ldots, t_n$.
- one pass over each input; heads may proceed asynchronously
- advancement of heads and new content of memory depends on the current content of memory and the symbols seen at both heads
- for simplicity: advancement of only one head at a time
- s : size of memory buffer (number of bits)
- m : number of possible memory configurations, i.e., $\log m = s$

How to prove lower bounds in this scenario?

Problem:
"Classical" communication complexity results cannot be used so easily here.

Solution: Take a direct look at the "flow of information" during computations.

Consider the following example:

- $n \geqslant 2$
- $\mathbb{D}_n := \{a_1, b_1, c_1 \ldots, a_n, b_n, c_n\}$ — domain of $3n$ input items
- variation of the set disjointness problem:

DISJ$_n$

Input: Two streams $S = s_1, s_2, \ldots, s_n$ and $T = t_1, t_2, \ldots, t_n$ of elements in $\mathbb{D}_n$

Question: Is $\{s_1, s_2, \ldots, s_n\} \cap \{t_1, t_2, \ldots, t_n\} = \emptyset$?

How to prove lower bounds in this scenario?

Problem:
"Classical" communication complexity results cannot be used so easily here.

Solution: Take a direct look at the "flow of information" during computations.

Consider the following example:

- $n \geqslant 2$
- $\mathbb{D}_n := \{a_1, b_1, c_1 \ldots, a_n, b_n, c_n\}$ — domain of $3n$ input items
- variation of the set disjointness problem:

DISJ$_n$

Input: Two streams $S = s_1, s_2, \ldots, s_n$ and $T = t_1, t_2, \ldots, t_n$ of elements in $\mathbb{D}_n$

Question: Is $\{s_1, s_2, \ldots, s_n\} \cap \{t_1, t_2, \ldots, t_n\} = \emptyset$?

How to prove lower bounds in this scenario?

Problem:
"Classical" communication complexity results cannot be used so easily here.

Solution: Take a direct look at the "flow of information" during computations.

Consider the following example:

- $n \geqslant 2$
- $\mathbb{D}_n := \{a_1, b_1, c_1 \ldots, a_n, b_n, c_n\}$ — domain of $3n$ input items
- variation of the set disjointness problem:

$\textsc{Disj}_n$
Input: Two streams $S = s_1, s_2, \ldots, s_n$ and $T = t_1, t_2, \ldots, t_n$ of elements in $\mathbb{D}_n$ such that $s_i \in \{a_i, b_i\}$ and $t_{n-i+1} \in \{a_i, c_i\}$
Question: Is $\{s_1, s_2, \ldots, s_n\} \cap \{t_1, t_2, \ldots, t_n\} = \emptyset$?

A lower bound proof for $\textsc{Disj}_n$ (1/5)

$\textsc{Disj}_n$ $\mathbb{D}_n := \{a_1, b_1, c_1 \ldots, a_n, b_n, c_n\}$

Input: two streams $S = s_1, s_2, \ldots, s_n$, $T = t_1, t_2, \ldots, t_n$ of elements in $\mathbb{D}_n$, such that $s_i \in \{a_i, b_i\}$ and $t_{n-i+1} \in \{a_i, c_i\}$.

Question: Is $\{s_1, s_2, \ldots, s_n\} \cap \{t_1, t_2, \ldots, t_n\} = \emptyset$?

Theorem: *(Bar Yossef, Shalem, ICDE'08)*

$\textsc{Disj}_n$ *cannot be solved by a deterministic algorithm that performs one pass over each stream and that uses less than* $n - \log n - 1$ *bits of memory.*

Proof:

- Consider input instances $D(I_1, I_2) := (S_{I_1}, T_{I_2})$ with $I_1, I_2 \subseteq \{1, \ldots, n\}$ and
  - S_{I_1} : $i \in I_1 \implies$ i-th position carries a_i
    $i \notin I_1 \implies$ i-th position carries b_i
  - T_{I_2} : $i \in I_2 \implies$ $(n-i+1)$-th position carries a_i
    $i \notin I_2 \implies$ $(n-i+1)$-th position carries c_i
- Note: $S_{I_1} \cap T_{I_2} = \emptyset \iff I_1 \cap I_2 = \emptyset$
- Restrict attention to input instances $D(I, \bar{I}) = (S_I, T_{\bar{I}})$ for $I \subseteq \{1, \ldots, n\}$.
  (particular "yes"-instances)

A lower bound proof for DISJ_n (1/5)

DISJ_n $\qquad\qquad \mathbb{D}_n := \{a_1, b_1, c_1 \ldots, a_n, b_n, c_n\}$

Input: two streams $S = s_1, s_2, \ldots, s_n$, $T = t_1, t_2, \ldots, t_n$ of elements in $\mathbb{D}_n$, such that $s_i \in \{a_i, b_i\}$ and $t_{n-i+1} \in \{a_i, c_i\}$.

Question: Is $\{s_1, s_2, \ldots, s_n\} \cap \{t_1, t_2, \ldots, t_n\} = \emptyset$?

Theorem: *(Bar Yossef, Shalem, ICDE'08)*

DISJ_n *cannot be solved by a deterministic algorithm that performs one pass over each stream and that uses less than* $n - \log n - 1$ *bits of memory.*

Proof:

- Consider input instances $D(I_1, I_2) := (S_{I_1}, T_{I_2})$ with $I_1, I_2 \subseteq \{1, \ldots, n\}$ and
  - S_{I_1} : $i \in I_1 \implies i$-th position carries a_i; $i \notin I_1 \implies i$-th position carries b_i
  - T_{I_2} : $i \in I_2 \implies (n-i+1)$-th position carries a_i; $i \notin I_2 \implies (n-i+1)$-th position carries c_i
- Note: $S_{I_1} \cap T_{I_2} = \emptyset \iff I_1 \cap I_2 = \emptyset$
- Restrict attention to input instances $D(I, \bar{I}) = (S_I, T_{\bar{I}})$ for $I \subseteq \{1, \ldots, n\}$.
  (particular "yes"-instances)

A lower bound proof for DISJ_n (1/5)

DISJ_n $\qquad\qquad$ $\mathbb{D}_n := \{a_1, b_1, c_1 \dots, a_n, b_n, c_n\}$

Input: two streams $S = s_1, s_2, \dots, s_n$, $T = t_1, t_2, \dots, t_n$ of elements in $\mathbb{D}_n$, such that $s_i \in \{a_i, b_i\}$ and $t_{n-i+1} \in \{a_i, c_i\}$.

Question: Is $\{s_1, s_2, \dots, s_n\} \cap \{t_1, t_2, \dots, t_n\} = \emptyset$?

Theorem: *(Bar Yossef, Shalem, ICDE'08)*

DISJ_n *cannot be solved by a deterministic algorithm that performs one pass over each stream and that uses less than* $n - \log n - 1$ *bits of memory.*

Proof:

- Consider input instances $D(I_1, I_2) := (S_{I_1}, T_{I_2})$ with $I_1, I_2 \subseteq \{1, \dots, n\}$ and
  - S_{I_1} : $i \in I_1 \implies i$-th position carries a_i
    $i \notin I_1 \implies i$-th position carries b_i
  - T_{I_2} : $i \in I_2 \implies (n-i+1)$-th position carries a_i
    $i \notin I_2 \implies (n-i+1)$-th position carries c_i
- Note: $S_{I_1} \cap T_{I_2} = \emptyset \iff I_1 \cap I_2 = \emptyset$
- Restrict attention to input instances $D(I, \bar{I}) = (S_I, T_{\bar{I}})$ for $I \subseteq \{1, \dots, n\}$.
  (particular "yes"-instances)

A lower bound proof for DISJ_n (1/5)

DISJ_n $\qquad$ $\mathbb{D}_n := \{a_1, b_1, c_1 \ldots, a_n, b_n, c_n\}$

Input: two streams $S = s_1, s_2, \ldots, s_n$, $T = t_1, t_2, \ldots, t_n$ of elements in $\mathbb{D}_n$, such that $s_i \in \{a_i, b_i\}$ and $t_{n-i+1} \in \{a_i, c_i\}$.

Question: Is $\{s_1, s_2, \ldots, s_n\} \cap \{t_1, t_2, \ldots, t_n\} = \emptyset$?

Theorem: *(Bar Yossef, Shalem, ICDE'08)*

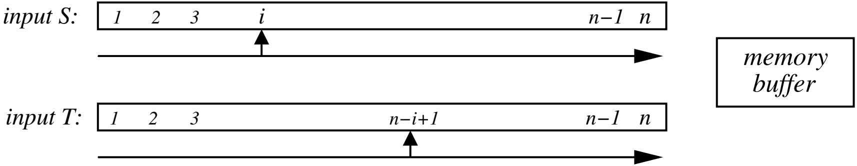
DISJ_n *cannot be solved by a deterministic algorithm that performs one pass over each stream and that uses less than* $n - \log n - 1$ *bits of memory.*

Proof:

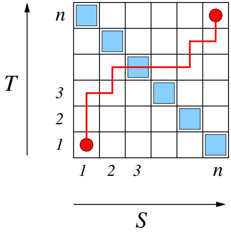
- Consider input instances $D(I_1, I_2) := (S_{I_1}, T_{I_2})$ with $I_1, I_2 \subseteq \{1, \ldots, n\}$ and
  - S_{I_1} : $i \in I_1 \implies$ i-th position carries a_i
    $i \notin I_1 \implies$ i-th position carries b_i
  - T_{I_2} : $i \in I_2 \implies$ $(n-i+1)$-th position carries a_i
    $i \notin I_2 \implies$ $(n-i+1)$-th position carries c_i
- Note: $S_{I_1} \cap T_{I_2} = \emptyset \iff I_1 \cap I_2 = \emptyset$
- Restrict attention to input instances $D(I, \bar{I}) = (S_I, T_{\bar{I}})$ for $I \subseteq \{1, \ldots, n\}$.
  (particular "yes"-instances)

A lower bound proof for $\textsc{Disj}_n$ (2/5)

Situation during a computation:



- potential head positions: (i, j) with $1 \leqslant i, j \leqslant n$
- start: $(1, 1)$
- end: (n, n)



- For each input $D(I, \bar{I})$ there exists exactly one $i \in \{1, \ldots, n\}$ such that the heads visit position $(i, n-i+1)$.

A lower bound proof for DISJ_n (3/5)

Goal now: "cut-and-paste argument"

Find $I, J \subseteq \{1, .., n\}$ such that computations on $D(I, \bar{I})$ and $D(J, \bar{J})$ can be combined to an accepting computation on $D(I', J')$ for I' and J' with $I' \cap J' \neq \emptyset$.

$\Longrightarrow$ accept a "no"-instance!

(1) Ex. $i \in \{1, .., n\}$ and $X_1 \subseteq \{I : I \subseteq \{1, .., n\}\}$ such that

- for each $I \in X_1$, head position $(i, n{-}i{+}1)$ is visited,
- $|X_1| \geqslant \frac{2^n}{n}$.

(2) Ex. $X_2 \subseteq X_1$ such that

- for all $I, J \in X_2$: $\quad i \in I \iff i \in J$,
- $|X_2| \geqslant \frac{|X_1|}{2} \geqslant \frac{2^n}{2n}$.

(3) Ex. memory configuration c and $X_3 \subseteq X_2$ such that

- for all $I \in X_3$: memory configuration c when at head position $(i, n{-}i{+}1)$,
- $|X_3| \geqslant \frac{|X_2|}{m} \geqslant \frac{2^n}{2nm}$.

Note: $\quad |X_3| > 1 \iff m < \frac{2^n}{2n} \iff s = \log m < n - \log n - 1$.

A lower bound proof for $\textsc{Disj}_n$ (3/5)

Goal now: "cut-and-paste argument"

Find $I, J \subseteq \{1,\ldots,n\}$ such that computations on $D(I,\bar{I})$ and $D(J,\bar{J})$ can be combined to an accepting computation on $D(I',J')$ for I' and J' with $I' \cap J' \neq \emptyset$.
$\Longrightarrow$ accept a "no"-instance!

(1) Ex. $i \in \{1,\ldots,n\}$ and $X_1 \subseteq \{I : I \subseteq \{1,\ldots,n\}\}$ such that
- for each $I \in X_1$, head position $(i, n{-}i{+}1)$ is visited,
- $|X_1| \geqslant \frac{2^n}{n}$.

(2) Ex. $X_2 \subseteq X_1$ such that
- for all $I, J \in X_2$: $\quad i \in I \iff i \in J$,
- $|X_2| \geqslant \frac{|X_1|}{2} \geqslant \frac{2^n}{2n}$.

(3) Ex. memory configuration c and $X_3 \subseteq X_2$ such that
- for all $I \in X_3$: memory configuration c when at head position $(i, n{-}i{+}1)$,
- $|X_3| \geqslant \frac{|X_2|}{m} \geqslant \frac{2^n}{2nm}$.

Note: $\quad |X_3| > 1 \iff m < \frac{2^n}{2n} \iff s = \log m < n - \log n - 1.$

A lower bound proof for DISJ$_n$ (3/5)

Goal now: "cut-and-paste argument"

Find $I, J \subseteq \{1,\ldots,n\}$ such that computations on $D(I,\bar{I})$ and $D(J,\bar{J})$ can be combined to an accepting computation on $D(I',J')$ for I' and J' with $I' \cap J' \neq \emptyset$.

$\Longrightarrow$ accept a "no"-instance!

(1) Ex. $i \in \{1,\ldots,n\}$ and $X_1 \subseteq \{I : I \subseteq \{1,\ldots,n\}\}$ such that

- for each $I \in X_1$, head position $(i, n{-}i{+}1)$ is visited,
- $|X_1| \geqslant \frac{2^n}{n}$.

(2) Ex. $X_2 \subseteq X_1$ such that

- for all $I, J \in X_2$: $\quad i \in I \iff i \in J$,
- $|X_2| \geqslant \frac{|X_1|}{2} \geqslant \frac{2^n}{2n}$.

(3) Ex. memory configuration c and $X_3 \subseteq X_2$ such that

- for all $I \in X_3$: memory configuration c when at head position $(i, n{-}i{+}1)$,
- $|X_3| \geqslant \frac{|X_2|}{m} \geqslant \frac{2^n}{2nm}$.

Note: $\quad |X_3| > 1 \iff m < \frac{2^n}{2n} \iff s = \log m < n - \log n - 1.$

A lower bound proof for DISJ$_n$ (3/5)

Goal now: "cut-and-paste argument"

Find $I, J \subseteq \{1, .., n\}$ such that computations on $D(I, \overline{I})$ and $D(J, \overline{J})$ can be combined to an accepting computation on $D(I', J')$ for I' and J' with $I' \cap J' \neq \emptyset$.

$\Longrightarrow$ accept a "no"-instance!

(1) Ex. $i \in \{1, .., n\}$ and $X_1 \subseteq \{I : I \subseteq \{1, .., n\}\}$ such that

- for each $I \in X_1$, head position $(i, n{-}i{+}1)$ is visited,
- $|X_1| \geqslant \frac{2^n}{n}$.

(2) Ex. $X_2 \subseteq X_1$ such that

- for all $I, J \in X_2$: $\quad i \in I \iff i \in J$,
- $|X_2| \geqslant \frac{|X_1|}{2} \geqslant \frac{2^n}{2n}$.

(3) Ex. memory configuration c and $X_3 \subseteq X_2$ such that

- for all $I \in X_3$: memory configuration c when at head position $(i, n{-}i{+}1)$,
- $|X_3| \geqslant \frac{|X_2|}{m} \geqslant \frac{2^n}{2nm}$.

Note: $\quad |X_3| > 1 \iff m < \frac{2^n}{2n} \iff s = \log m < n - \log n - 1.$

A lower bound proof for DISJ_n (3/5)

Goal now: "cut-and-paste argument"

Find $I, J \subseteq \{1, .., n\}$ such that computations on $D(I, \bar{I})$ and $D(J, \bar{J})$ can be combined to an accepting computation on $D(I', J')$ for I' and J' with $I' \cap J' \neq \emptyset$.

$\Longrightarrow$ accept a "no"-instance!

(1) Ex. $i \in \{1, .., n\}$ and $X_1 \subseteq \{I : I \subseteq \{1, .., n\}\}$ such that

- for each $I \in X_1$, head position $(i, n-i+1)$ is visited,
- $|X_1| \geqslant \frac{2^n}{n}$.

(2) Ex. $X_2 \subseteq X_1$ such that

- for all $I, J \in X_2$: $\quad i \in I \iff i \in J$,
- $|X_2| \geqslant \frac{|X_1|}{2} \geqslant \frac{2^n}{2n}$.

(3) Ex. memory configuration c and $X_3 \subseteq X_2$ such that

- for all $I \in X_3$: memory configuration c when at head position $(i, n-i+1)$,
- $|X_3| \geqslant \frac{|X_2|}{m} \geqslant \frac{2^n}{2nm}$.

Note: $\quad |X_3| > 1 \iff m < \frac{2^n}{2n} \iff s = \log m < n - \log n - 1.$

A lower bound proof for DISJ_n (3/5)

Goal now: "cut-and-paste argument"

Find $I, J \subseteq \{1, \ldots, n\}$ such that computations on $D(I, \overline{I})$ and $D(J, \overline{J})$ can be combined to an accepting computation on $D(I', J')$ for I' and J' with $I' \cap J' \neq \emptyset$.

$\Longrightarrow$ accept a "no"-instance!

(1) Ex. $i \in \{1, \ldots, n\}$ and $X_1 \subseteq \{I : I \subseteq \{1, \ldots, n\}\}$ such that
- for each $I \in X_1$, head position $(i, n-i+1)$ is visited,
- $|X_1| \geqslant \frac{2^n}{n}$.

(2) Ex. $X_2 \subseteq X_1$ such that
- for all $I, J \in X_2$: $\quad i \in I \iff i \in J$,
- $|X_2| \geqslant \frac{|X_1|}{2} \geqslant \frac{2^n}{2n}$.

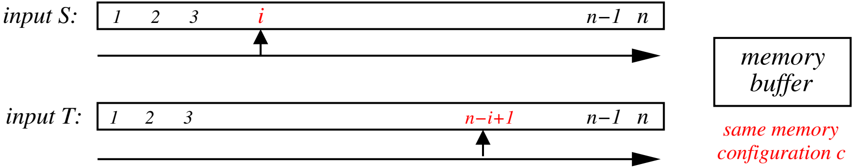
(3) Ex. memory configuration c and $X_3 \subseteq X_2$ such that
- for all $I \in X_3$: memory configuration c when at head position $(i, n-i+1)$,
- $|X_3| \geqslant \frac{|X_2|}{m} \geqslant \frac{2^n}{2nm}$.

Note: $\quad |X_3| > 1 \quad \Longleftarrow \quad m < \frac{2^n}{2n} \quad \iff \quad s = \log m < n - \log n - 1.$

A lower bound proof for DISJ_n (4/5)

Let $I, J \in X_3$ with $I \neq J$.

Same situation on input $D(I, \bar{I})$ and on input $D(J, \bar{J})$:



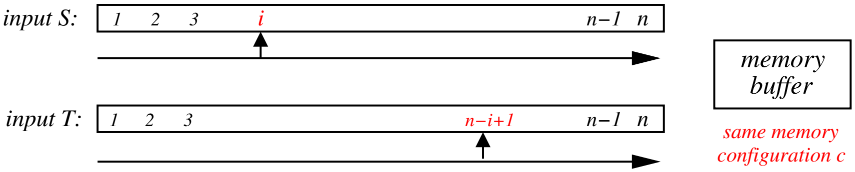
- Cut-and-paste argument $\Longrightarrow$ Same situation on inputs $D(I_1, I_2)$ and $D(I_1', I_2')$
- $$\begin{aligned} I_1 &= (I \cap \{1,\ldots,i-1\}) \cup (I \cap \{i\}) \cup (J \cap \{i+1,\ldots,n\}) \\ I_2 &= (\bar{I} \cap \{i+1,\ldots,n\}) \cup (\bar{I} \cap \{i\}) \cup (\bar{J} \cap \{1,\ldots,i-1\}) \end{aligned}$$
- $$\begin{aligned} I_1' &= (J \cap \{1,\ldots,i-1\}) \cup (J \cap \{i\}) \cup (I \cap \{i+1,\ldots,n\}) \\ I_2' &= (\bar{J} \cap \{i+1,\ldots,n\}) \cup (\bar{J} \cap \{i\}) \cup (\bar{I} \cap \{1,\ldots,i-1\}) \end{aligned}$$

Since $I \neq J$, $D(I_1, I_2)$ or $D(I_1', I_2')$ is a "no"-instance. $\square$

A lower bound proof for $\textsc{Disj}_n$ (4/5)

Let $I, J \in X_3$ with $I \neq J$.

Same situation on input $D(I, \bar{I})$ and on input $D(J, \bar{J})$:



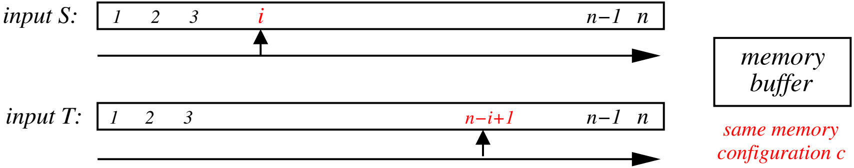
- Cut-and-paste argument $\Longrightarrow$ Same situation on inputs $D(I_1, I_2)$ and $D(I_1', I_2')$
- $$I_1 = (I \cap \{1, .., i-1\}) \cup (I \cap \{i\}) \cup (J \cap \{i+1, .., n\})$$
  $$I_2 = (\bar{I} \cap \{i+1, .., n\}) \cup (\bar{I} \cap \{i\}) \cup (\bar{J} \cap \{1, .., i-1\})$$
- $$I_1' = (J \cap \{1, .., i-1\}) \cup (J \cap \{i\}) \cup (I \cap \{i+1, .., n\})$$
  $$I_2' = (\bar{J} \cap \{i+1, .., n\}) \cup (\bar{J} \cap \{i\}) \cup (\bar{I} \cap \{1, .., i-1\})$$

Since $I \neq J$, $D(I_1, I_2)$ or $D(I_1', I_2')$ is a "no"-instance.

A lower bound proof for $\textsc{Disj}_n$ (4/5)

Let $I, J \in X_3$ with $I \neq J$.

Same situation on input $D(I, \bar{I})$ and on input $D(J, \bar{J})$:



- Cut-and-paste argument $\Longrightarrow$ Same situation on inputs $D(I_1, I_2)$ and $D(I_1', I_2')$
- $$I_1 = (I \cap \{1,..,i-1\}) \cup (I \cap \{i\}) \cup (J \cap \{i+1,..,n\})$$
  $$I_2 = (\bar{I} \cap \{i+1,..,n\}) \cup (\bar{I} \cap \{i\}) \cup (\bar{J} \cap \{1,..,i-1\})$$
- $$I_1' = (J \cap \{1,..,i-1\}) \cup (J \cap \{i\}) \cup (I \cap \{i+1,..,n\})$$
  $$I_2' = (\bar{J} \cap \{i+1,..,n\}) \cup (\bar{J} \cap \{i\}) \cup (\bar{I} \cap \{1,..,i-1\})$$

Since $I \neq J$, $D(I_1, I_2)$ or $D(I_1', I_2')$ is a "no"-instance. $\square$

A lower bound proof for DISJ_n (5/5)

We have proved

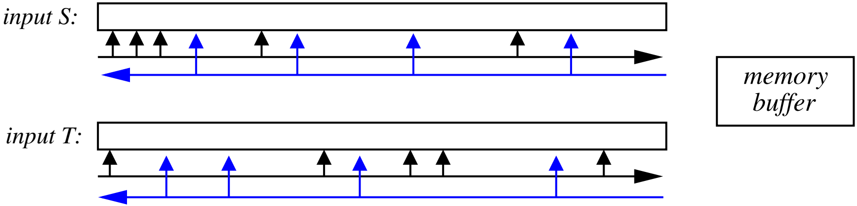
Theorem: *(Bar Yossef, Shalem, ICDE'08)*

DISJ_n *cannot be solved by a deterministic algorithm that performs one pass over each stream and that uses less than* $n - \log n - 1$ *bits of memory.*

The proof given by Bar-Yossef and Shalem (ICDE 2008) is different.
For their proof, they introduce a particular kind of communication model: the token-based mesh communication model.

Several passes over several streams in parallel

General scenario: mp2s-automaton $\mathcal{A}$ with parameters $(\mathbb{D}, m, k_f, k_b)$

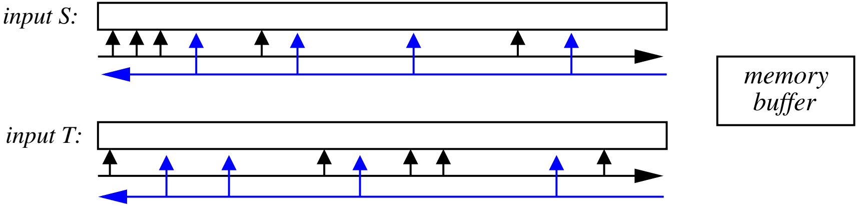


Parameters:

- 2 input streams: $S = s_1, s_2, \ldots, s_n$ and $T = t_1, t_2, \ldots, t_n$ of elements in $\mathbb{D}$.
- m : number of possible memory configurations; $s := \log m$ size of the memory buffer (number of bits).
- k_f forward heads on each input stream, k_b backward heads on each input stream
- Depending on (a) the current memory state and (b) the elements in S and T at the current head positions, a deterministic transition function determines (1) the next memory state and (2) which of the heads should be advanced to the next position.

Several passes over several streams in parallel

General scenario: mp2s-automaton $\mathcal{A}$ with parameters $(\mathbb{D}, m, k_f, k_b)$

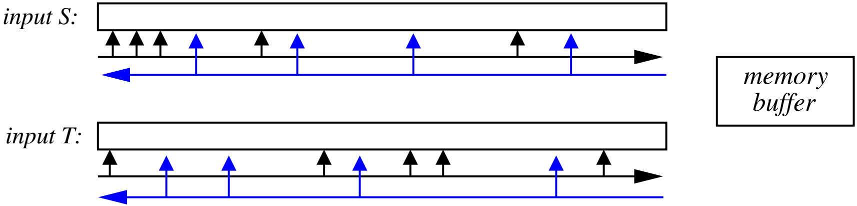


Parameters:

- 2 input streams: $S = s_1, s_2, \ldots, s_n$ and $T = t_1, t_2, \ldots, t_n$ of elements in $\mathbb{D}$.
- m : number of possible memory configurations;
  $s := \log m$ size of the memory buffer (number of bits).
- k_f forward heads on each input stream,
  k_b backward heads on each input stream
- Depending on (a) the current memory state and (b) the elements in S and T at the current head positions, a deterministic transition function determines (1) the next memory state and (2) which of the heads should be advanced to the next position.

Several passes over several streams in parallel

General scenario: mp2s-automaton $\mathcal{A}$ with parameters $(\mathbb{D}, m, k_f, k_b)$



Parameters:

- 2 input streams: $S = s_1, s_2, \ldots, s_n$ and $T = t_1, t_2, \ldots, t_n$ of elements in $\mathbb{D}$.
- m : number of possible memory configurations;
  $s := \log m$ size of the memory buffer (number of bits).
- k_f forward heads on each input stream,
  k_b backward heads on each input stream
- Depending on (a) the current memory state and (b) the elements in S and T at the current head positions, a deterministic transition function determines (1) the next memory state and (2) which of the heads should be advanced to the next position.

Solving $\textsc{Disj}_n$ with an mp2s-automaton: upper bound

Proposition:

$\textsc{Disj}_n$ *can be solved by an mp2s-automaton with parameters* $(\mathbb{D}_n, n{+}2, \sqrt{n}, 0)$.
(I.e.: memory buffer of $\log(n{+}2)$ *bits,* $\sqrt{n}$ *forward heads, no backward heads)*

Proof:

Solving DISJ_n with an mp2s-automaton: upper bound

Proposition:

DISJ_n *can be solved by an mp2s-automaton with parameters* $(\mathbb{D}_n, n{+}2, \sqrt{n}, 0)$.
(I.e.: memory buffer of $\log(n{+}2)$ *bits,* $\sqrt{n}$ *forward heads, no backward heads)*

Proof:

Phase 1:

Move heads on S such that they partition S into blocks of length $\sqrt{n}$.
(use $n{+}1 - \sqrt{n}$ states)

Phase 2:

For $j = 1, \ldots, \sqrt{n}$ do

(1) Let j-th head on T pass the entire stream and compare each element of T with the $\sqrt{n}$ elements at head positions in S.

(2) Advance each head on S one step to the right.

(use 2 states)

Solving $\textsc{Disj}_n$ with an mp2s-automaton: upper bound

Proposition:

$\textsc{Disj}_n$ *can be solved by an mp2s-automaton with parameters* $(\mathbb{D}_n, n{+}2, \sqrt{n}, 0)$.
(I.e.: memory buffer of $\log(n{+}2)$ *bits,* $\sqrt{n}$ *forward heads, no backward heads)*

Proof:

Phase 1:

Move heads on S such that they partition S into blocks of length $\sqrt{n}$.

(use $n{+}1 - \sqrt{n}$ states)

Phase 2:

For $j = 1, \ldots, \sqrt{n}$ do

(1) Let j-th head on T pass the entire stream and compare each element of T with the $\sqrt{n}$ elements at head positions in S.

(2) Advance each head on S one step to the right.

(use 2 states)

Solving DISJ_n with an mp2s-automaton: lower bound

Theorem: *(S, STACS'09)*

For all n, m, k_f, k_b such that, for $k = 2k_f + 2k_b$ and $v = (k_f^2 + k_b^2 + 1)\cdot(2k_f k_b + 1)$,

$$k^2 \cdot v \cdot \log(n+1) \; + \; k \cdot v \cdot \log m \; + \; v \cdot (1 + \lg v) \quad \leqslant \quad n,$$

the problem DISJ_n cannot be solved by any mp2s-automaton with parameters $(\mathbb{D}_n, m, k_f, k_b)$.

Proof:

- Similar to the shown proof where only one forward head is available on each stream.
- Divide input streams into blocks and choose a block that is “not checked” by any pair of cursors.

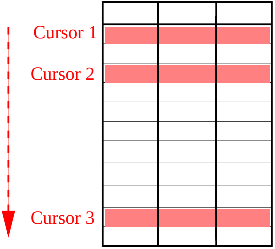
Finite Cursor Machines

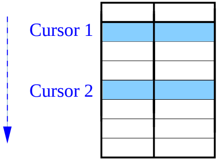
Introduced by Grohe, Gurevich, Leinders, S., Tyszkiewicz, Van den Bussche, ICDT'07

- an abstract model for database query processing
- formal model: based on Abstract State Machines

Informal Description of a FCM:

- works on a relational database
  (tables, not sets) (read-only access)
- on each table:
  a fixed number of cursors
- cursors are one-way,
  but can move asynchronously
- internal memory:
  - finite state control
  - fixed number of registers which
    can store bitstrings
- manipulation of output row and internal
  memory: via built-in bitstring functions
  on data elements and bitstrings





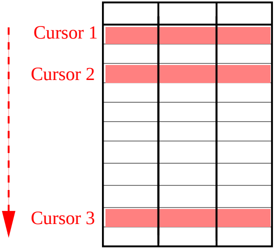
Finite Cursor Machines

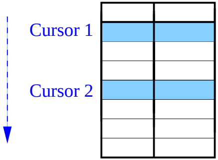
Introduced by Grohe, Gurevich, Leinders, S., Tyszkiewicz, Van den Bussche, ICDT'07

- an abstract model for database query processing
- formal model: based on Abstract State Machines

Informal Description of a FCM:

- works on a relational database
  (tables, not sets) (read-only access)
- on each table:
  a fixed number of cursors
- cursors are one-way,
  but can move asynchronously
- internal memory:
  - finite state control
  - fixed number of registers which
    can store bitstrings
- manipulation of output row and internal
  memory: via built-in bitstring functions
  on data elements and bitstrings





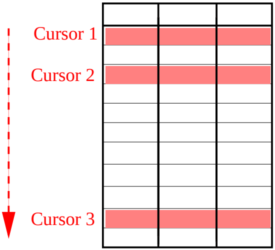
Finite Cursor Machines

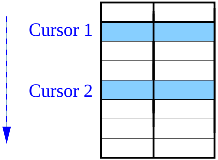
Introduced by Grohe, Gurevich, Leinders, S., Tyszkiewicz, Van den Bussche, ICDT'07

- an abstract model for database query processing
- formal model: based on Abstract State Machines

Informal Description of a FCM:

- works on a relational database
  (tables, not sets) (read-only access)
- on each table:
  a fixed number of cursors
- cursors are one-way,
  but can move asynchronously
- internal memory:
  - finite state control
  - fixed number of registers which can store bitstrings
- manipulation of output row and internal memory: via built-in bitstring functions on data elements and bitstrings





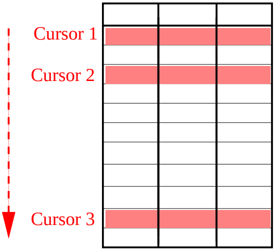
Finite Cursor Machines

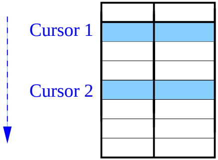
Introduced by Grohe, Gurevich, Leinders, S., Tyszkiewicz, Van den Bussche, ICDT'07

- an abstract model for database query processing
- formal model: based on Abstract State Machines

Informal Description of a FCM:

- works on a relational database
  (tables, not sets) (read-only access)
- on each table:
  a fixed number of cursors
- cursors are one-way,
  but can move asynchronously
- internal memory:
  - finite state control
  - fixed number of registers which
    can store bitstrings
- manipulation of output row and internal
  memory: via built-in bitstring functions
  on data elements and bitstrings





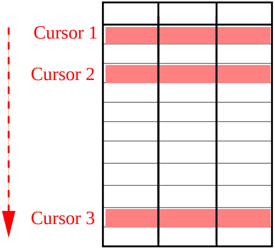
Finite Cursor Machines

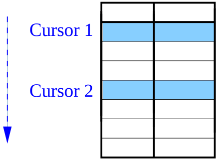
Introduced by Grohe, Gurevich, Leinders, S., Tyszkiewicz, Van den Bussche, ICDT'07

- an abstract model for database query processing
- formal model: based on Abstract State Machines

Informal Description of a FCM:

- works on a relational database
  (tables, not sets) (read-only access)
- on each table:
  a fixed number of cursors
- cursors are one-way,
  but can move asynchronously
- internal memory:
  - finite state control
  - fixed number of registers which
    can store bitstrings
- manipulation of output row and internal memory: via built-in bitstring functions on data elements and bitstrings





Easy Observations

Consider the operators from Relational Algebra

- Selection $\sigma_{i=j}(R)$ can be implemented by a FCM
- Union $R_1 \cup R_2$ and Projection $\pi_J(R)$ can be implemented by a FCM, provided that input tables are ordered
- Joins are NOT computable by FCMs, because the output size of a join can be quadratic, and FCMs can output only a linear number of different tuples
- Window Joins for a fixed window size w can be computed by an FCM (which has w cursors on each relation)
- Semijoins $R \ltimes_\theta S$ can be computed by an FCM, provided that input tables are ordered $R \ltimes_\theta S := \{t \in R : \text{there is an } s \in S \text{ such that } \theta(t, s)\}$

Corollary:

Each Semijoin Algebra query can be computed by query plan composed of FCMs and sorting operations. (a.k.a: "classical" 2-pass query processing)

Question: Are intermediate sorting steps really necessary?

Easy Observations

Consider the operators from Relational Algebra

- Selection $\sigma_{i=j}(R)$ can be implemented by a FCM
- Union $R_1 \cup R_2$ and Projection $\pi_J(R)$ can be implemented by a FCM, provided that input tables are ordered
- Joins are NOT computable by FCMs, because the output size of a join can be quadratic, and FCMs can output only a linear number of different tuples
- Window Joins for a fixed window size w can be computed by an FCM (which has w cursors on each relation)
- Semijoins $R \ltimes_\theta S$ can be computed by an FCM, provided that input tables are ordered $R \ltimes_\theta S := \{t \in R : \text{there is an } s \in S \text{ such that } \theta(t,s)\}$

Corollary:

Each Semijoin Algebra query can be computed by query plan composed of FCMs and sorting operations. *(a.k.a: "classical" 2-pass query processing)*

Question: Are intermediate sorting steps really necessary?

Easy Observations

Consider the operators from Relational Algebra

- Selection $\sigma_{i=j}(R)$ can be implemented by a FCM
- Union $R_1 \cup R_2$ and Projection $\pi_J(R)$ can be implemented by a FCM, provided that input tables are ordered
- Joins are NOT computable by FCMs, because the output size of a join can be quadratic, and FCMs can output only a linear number of different tuples
- Window Joins for a fixed window size w can be computed by an FCM (which has w cursors on each relation)
- Semijoins $R \ltimes_\theta S$ can be computed by an FCM, provided that input tables are ordered $R \ltimes_\theta S := \{t \in R : \text{there is an } s \in S \text{ such that } \theta(t,s)\}$

Corollary:

Each Semijoin Algebra query can be computed by query plan composed of FCMs and sorting operations. *(a.k.a: "classical" 2-pass query processing)*

Question: Are intermediate sorting steps really necessary?

Easy Observations

Consider the operators from Relational Algebra

- Selection $\sigma_{i=j}(R)$ can be implemented by a FCM
- Union $R_1 \cup R_2$ and Projection $\pi_J(R)$ can be implemented by a FCM, provided that input tables are ordered
- Joins are NOT computable by FCMs, because the output size of a join can be quadratic, and FCMs can output only a linear number of different tuples
- Window Joins for a fixed window size w can be computed by an FCM (which has w cursors on each relation)
- Semijoins $R \ltimes_\theta S$ can be computed by an FCM, provided that input tables are ordered $R \ltimes_\theta S := \{t \in R : \text{there is an } s \in S \text{ such that } \theta(t,s)\}$

Corollary:

Each Semijoin Algebra query can be computed by query plan composed of FCMs and sorting operations. *(a.k.a: "classical" 2-pass query processing)*

Question: Are intermediate sorting steps really necessary?

Easy Observations

Consider the operators from Relational Algebra

- Selection $\sigma_{i=j}(R)$ can be implemented by a FCM
- Union $R_1 \cup R_2$ and Projection $\pi_J(R)$ can be implemented by a FCM, provided that input tables are ordered
- Joins are NOT computable by FCMs, because the output size of a join can be quadratic, and FCMs can output only a linear number of different tuples
- Window Joins for a fixed window size w can be computed by an FCM (which has w cursors on each relation)
- Semijoins $R \ltimes_\theta S$ can be computed by an FCM, provided that input tables are ordered $R \ltimes_\theta S := \{t \in R : \text{there is an } s \in S \text{ such that } \theta(t,s)\}$

Corollary:

Each Semijoin Algebra query can be computed by query plan composed of FCMs and sorting operations. *(a.k.a: "classical" 2-pass query processing)*

Question: Are intermediate sorting steps really necessary?

Easy Observations

Consider the operators from Relational Algebra

- Selection $\sigma_{i=j}(R)$ can be implemented by a FCM
- Union $R_1 \cup R_2$ and Projection $\pi_J(R)$ can be implemented by a FCM, provided that input tables are ordered
- Joins are NOT computable by FCMs, because the output size of a join can be quadratic, and FCMs can output only a linear number of different tuples
- Window Joins for a fixed window size w can be computed by an FCM (which has w cursors on each relation)
- Semijoins $R \ltimes_\theta S$ can be computed by an FCM, provided that input tables are ordered $R \ltimes_\theta S := \{t \in R : \text{there is an } s \in S \text{ such that } \theta(t, s)\}$

Corollary:

Each Semijoin Algebra query can be computed by query plan composed of FCMs and sorting operations. *(a.k.a: "classical" 2-pass query processing)*

Question: Are intermediate sorting steps really necessary?

Question:
Are intermediate sorting steps really necessary?

Answer: Yes! ...

Theorem: *(Grohe, Gurevich, Leinders, S., Tyszkiewicz, Van den Bussche, ICDT'07)*

The query

$$\text{Is} \quad R \ltimes_{x_1=y_1} (S \ltimes_{x_2=y_1} T) \quad \text{nonempty?}$$

where R and T are unary and S in binary, is not computable by an FCM (even if the FCM is allowed to have as input all sorted versions of the input relations).

An Open Question

Is there a Boolean query from Relational Algebra (or, equivalently, a sentence of first-order logic), that cannot be computed by any composition of FCMs and sorting operations?

Conjecture: Yes

... since otherwise FO would have data complexity of time $n \cdot \log n$

An Open Question

Is there a Boolean query from Relational Algebra (or, equivalently, a sentence of first-order logic), that cannot be computed by any composition of FCMs and sorting operations?

Conjecture: Yes

… since otherwise FO would have data complexity of time $n \cdot \log n$

Overview

One pass over a single stream

Several passes over a single stream

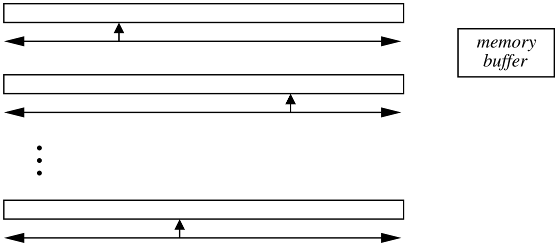
Several passes over several streams in parallel

Read/write streams

Future tasks

Read/write streams

Scenario:



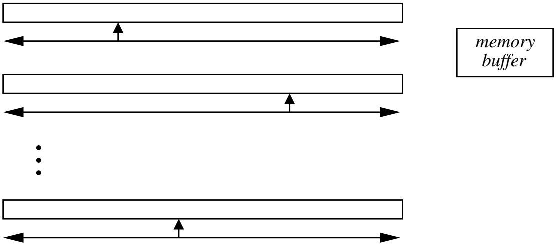
Parameters:

- t read/write streams
- one head on each stream; each head can write onto (and append) the stream
- r : maximum number of head reversals
- s : size of "internal memory" (number of bits)

- input on first read/write stream
- if necessary: output on last read/write stream
- formal model: based on Turing machines.

Read/write streams

Scenario:



Parameters:

- t read/write streams
- one head on each stream; each head can write onto (and append) the stream
- r : maximum number of head reversals
- s : size of "internal memory" (number of bits)

- input on first read/write stream
- if necessary: output on last read/write stream
- formal model: based on Turing machines.

Complexity classes

$\mathrm{ST}(r, s, t)$:

class of all problems that can be solved by a deterministic algorithm using

- t read/write streams,
- at most r head reversals, and
- a memory buffer of size s.

The sorting problem

SORTING — *Input length* $N = m \cdot (n+1)$

Input: bit-strings $x_1, \ldots, x_m \in \{0,1\}^n$ (for arbitrary m, n)

Output: $x_1, \ldots, x_m$ sorted in ascending order

Already seen in this talk :

Theorem: *(Grohe, Koch, S., ICALP'05)*

SORTING *can be solved by a* (p, s)*-bounded computation* $\iff$ $(p \cdot s) \in \Omega(N)$

Thus: SORTING $\in \mathrm{ST}(r, s, 1) \iff r(N) \cdot s(N) \in \Omega(N)$.

Theorem: *(Chen, Yap, 1991)*

SORTING $\in \mathrm{ST}(O(\log N), O(1), 2)$

Proof method: refinement of Merge-Sort.

The sorting problem

SORTING *Input length* $N = m \cdot (n+1)$

Input: bit-strings $x_1, \ldots, x_m \in \{0,1\}^n$ (for arbitrary m, n)

Output: $x_1, \ldots, x_m$ sorted in ascending order

Already seen in this talk :

Theorem: *(Grohe, Koch, S., ICALP'05)*

SORTING *can be solved by a* (p,s)*-bounded computation* $\iff$ $(p \cdot s) \in \Omega(N)$

Thus: SORTING $\in \mathrm{ST}(r, s, 1) \iff r(N) \cdot s(N) \in \Omega(N)$.

Theorem: *(Chen, Yap, 1991)*

SORTING $\in \mathrm{ST}(O(\log N), O(1), 2)$

Proof method: refinement of Merge-Sort.

Lower bound for sorting with $\geqslant 2$ r/w streams

Problem:

An additional read/write stream can be used to move around large parts of the input (with just 2 head reversals).

$\rightsquigarrow$ communication complexity does not help to prove lower bounds

Intuition:

Still, the order of the input strings cannot be changed so easily.

Fact:

For sufficiently small $r(N)$, $s(N)$, even with $t \geqslant 2$ read/write streams, sorting by solely comparing and moving around the input strings is impossible.

(For comparison-exchange algorithms, according lower bounds are well-known.)

Lower bound for sorting with $\geqslant 2$ r/w streams

Problem:

An additional read/write stream can be used to move around large parts of the input (with just 2 head reversals).

$\rightsquigarrow$ communication complexity does not help to prove lower bounds

Intuition:

Still, the order of the input strings cannot be changed so easily.

Fact:

For sufficiently small $r(N)$, $s(N)$, even with $t \geqslant 2$ read/write streams,
sorting by solely comparing and moving around the input strings is impossible.

(For comparison-exchange algorithms, according lower bounds are well-known.)

Lower bound for sorting with $\geqslant 2$ r/w streams

Problem:

Algorithms for read/write streams are based on Turing machines.
They can perform much more complicated operations than just compare and move around input strings.

Example:

During a first scan of the input, compute the sum of the input numbers modulo a large prime.
(In this way, already a single scan suffices to produce a number that depends in a non-trivial way on the entire input.)

⋮

Do some magic!

— Recall the data stream algorithms for MISSING NUMBER or MULTISET-EQUALITY !

⋮

Write the sorted sequence onto the output read/write stream.

Lower Bound for Sorting

Theorem: *(Grohe, S., PODS'05)*

SORTING $\notin \mathrm{ST}\big(o(\log N), N^{1-\varepsilon}, O(1)\big)$ *(for every $\varepsilon > 0$)*

Proof method:

1. New machine model: List Machines
   - can only compare and move around input strings ($\rightsquigarrow$ weaker than TMs)
   - non-uniform & lots of states and tape symbols ($\rightsquigarrow$ stronger than TMs)
2. Show that list machines can simulate algorithms on read/write streams.
3. Prove that list machines cannot sort (. . . use combinatorics).

Randomised ST-Classes: RST and co-RST

Definition of RST: analogous to the class RP (randomised polynomial time):

An RST-algorithm produces

- no "false positives", i.e., it rejects "no"-instances with prob. 1
- "false negatives" with prob. < 0.1, i.e. it accepts "yes"-inst. with prob. > 0.9

A co-RST-algorithm has complementary probabilities for accepting resp. rejecting:

- no "false negatives", i.e. it accepts "yes"-instances with prob. 1
- "false positives" with prob. < 0.1, i.e. it rejects "no"-inst. with prob. > 0.9

Theorem: *(Grohe, Hernich, S., PODS'06)*

MULTISET-EQUALITY
$$\begin{cases} \notin \text{RST}(o(\log N), N^{1-\varepsilon}, O(1)) & \textit{(for every } \varepsilon > 0) \\ \in \text{co-RST}(2, O(\log N), 1) & \\ \in \text{ST}(O(\log N), O(1), 2) & \end{cases}$$

Randomised ST-Classes: RST and co-RST

Definition of RST: analogous to the class RP (randomised polynomial time):

An RST-algorithm produces

- no "false positives", i.e., it rejects "no"-instances with prob. 1
- "false negatives" with prob. < 0.1, i.e. it accepts "yes"-inst. with prob. > 0.9

A co-RST-algorithm has complementary probabilities for accepting resp. rejecting:

- no "false negatives", i.e. it accepts "yes"-instances with prob. 1
- "false positives" with prob. < 0.1, i.e. it rejects "no"-inst. with prob. > 0.9

Theorem: *(Grohe, Hernich, S., PODS'06)*

$$\text{MULTISET-EQUALITY} \begin{cases} \notin \text{RST}(o(\log N), N^{1-\varepsilon}, O(1)) & \text{(for every } \varepsilon > 0\text{)} \\ \in \text{co-RST}(2, O(\log N), 1) \\ \in \text{ST}(O(\log N), O(1), 2) \end{cases}$$

Randomised ST-Classes: RST and co-RST

Definition of RST: analogous to the class RP (randomised polynomial time):

An RST-algorithm produces

- no "false positives", i.e., it rejects "no"-instances with prob. 1
- "false negatives" with prob. < 0.1, i.e. it accepts "yes"-inst. with prob. > 0.9

A co-RST-algorithm has complementary probabilities for accepting resp. rejecting:

- no "false negatives", i.e. it accepts "yes"-instances with prob. 1
- "false positives" with prob. < 0.1, i.e. it rejects "no"-inst. with prob. > 0.9

Theorem: *(Grohe, Hernich, S., PODS'06)*

$$\text{MULTISET-EQUALITY} \begin{cases} \notin \text{RST}(o(\log N), N^{1-\varepsilon}, O(1)) & \textit{(for every } \varepsilon > 0\textit{)} \\ \in \text{co-RST}(2, O(\log N), 1) \\ \in \text{ST}(O(\log N), O(1), 2) \end{cases}$$

Consequences

Separation of deterministic, randomised, and nondeterministic $\mathrm{ST}(\cdots)$-classes:

$$\begin{array}{ll} \mathrm{NST}(R, S, O(1)) & \\ \quad\quad | & \leftarrow \textsc{Multiset-Equality} \in \mathrm{NST}(3, O(\log N), 2) \\ \mathrm{RST}(R, S, O(1)) & \\ \quad\quad | & \leftarrow \textsc{Multiset-Equality} \in \text{co-}\mathrm{RST}(2, O(\log N), 1) \\ \mathrm{ST}(R, S, O(1)) & \end{array}$$

for all $R \subseteq o(\log n)$ and $O(\log n) \subseteq S \subseteq O(N^{1-\varepsilon})$

ST-Classes with 2-Sided Bounded Error

Definition of BPST: analogous to the class BPP (two-sided bounded error probabilistic polynomial time):

An BPST-machine produces

- "false positives" with prob. < 0.1, i.e., it rejects "no"-instances with prob. > 0.9
- "false negatives" with prob. < 0.1, it accepts "yes"-instances with prob. > 0.9

Theorem: *(Beame, Jayram, Rudra, STOC'07)*

SET-DISJOINTNESS $\notin$ BPST$\left(o\left(\frac{\log N}{\log\log N}\right), N^{1-\varepsilon}, O(1)\right)$ *(for every $\varepsilon > 0$)*

Theorem: *(Beame, Huynh-Ngoc, FOCS'08)*

Approximating the frequency moments F_k with a randomised read/write stream algorithm with $o(\log N)$ head reversals requires (almost) as much internal memory as a "conventional" one-pass data stream algorithm.

ST-Classes with 2-Sided Bounded Error

Definition of BPST: analogous to the class BPP (two-sided bounded error probabilistic polynomial time):

An BPST-machine produces

- "false positives" with prob. < 0.1, i.e., it rejects "no"-instances with prob. > 0.9
- "false negatives" with prob. < 0.1, it accepts "yes"-instances with prob. > 0.9

Theorem: *(Beame, Jayram, Rudra, STOC'07)*

SET-DISJOINTNESS $\notin$ BPST$\left(o\left(\frac{\log N}{\log\log N}\right), N^{1-\varepsilon}, O(1)\right)$ *(for every $\varepsilon > 0$)*

Theorem: *(Beame, Huynh-Ngoc, FOCS'08)*

Approximating the frequency moments F_k with a randomised read/write stream algorithm with $o(\log N)$ head reversals requires (almost) as much internal memory as a "conventional" one-pass data stream algorithm.

ST-Classes with 2-Sided Bounded Error

Definition of BPST: analogous to the class BPP (two-sided bounded error probabilistic polynomial time):

An BPST-machine produces

- "false positives" with prob. < 0.1, i.e., it rejects "no"-instances with prob. > 0.9
- "false negatives" with prob. < 0.1, it accepts "yes"-instances with prob. > 0.9

Theorem: *(Beame, Jayram, Rudra, STOC'07)*

SET-DISJOINTNESS $\notin$ BPST $\left(o\left(\frac{\log N}{\log\log N}\right), N^{1-\varepsilon}, O(1)\right)$ *(for every $\varepsilon > 0$)*

Theorem: *(Beame, Huynh-Ngoc, FOCS'08)*

Approximating the frequency moments F_k with a randomised read/write stream algorithm with $o(\log N)$ head reversals requires (almost) as much internal memory as a "conventional" one-pass data stream algorithm.

Overview

One pass over a single stream

Several passes over a single stream

Several passes over several streams in parallel

Read/write streams

Future tasks

Overview

One pass over a single stream

Several passes over a single stream

Several passes over several streams in parallel

Read/write streams

Future tasks

A few directions for future research

- Consider randomized versions of mp2s-automata:

  Design efficient randomized approximation algorithms for particular problems and develop techniques for proving lower bounds in the randomized model.

- Study the extension of the read/write stream model in which intermediate sorting steps are available.

  This is the StrSort model by Aggarwal, Datar, Rajagopalan, Ruhl, FOCS'04.

- An open question concerning finite cursor machines:

  Is there a sentence from first-order logic that cannot be evaluated by a composition of finite cursor machines and sorting operations?
  (Conjecture: yes!)

- An open question from complexity theory:

  Can the sorting problem be solved by a linear time multi-tape Turing machine?

Data stream talks during DEIS'10

- Data stream management systems and query languages (Tuesday, 8:45–9:45) — Sandra Geisler
- Basic algorithmic techniques for processing data streams (Tuesday, 9:45–10:45) — Mariano Zelke
- Querying and mining data streams (Wednesday, 11:15–12:15) — Elena Ikonomovska
- Stream-based processing of XML documents (Thursday, 11:15–12:15) — Cristian Riveros
- Distributed processing of data streams and large data sets (Thursday 1:45–2:45) — Marwan Hassani

Exercise # 4

Let s be a number with $0 < s < 1$.
The goal is to find a data stream algorithm that processes an input stream

$$x_1,\ x_2,\ x_3,\ \ldots,\ x_n$$

of elements from $\{1, \ldots, m\}$ and outputs a set M of input elements such that M contains (at least) all those elements that occur for $\geqslant s \cdot n$ times in the input stream.

Note:

- The output has to be a set — i.e., it is not allowed to output elements more than once. (In particular, this means that you cannot simply output the entire input stream.)
- The problem can be solved by a deterministic data stream algorithm using $O(\frac{1}{s} \cdot \log m \cdot \log n)$ memory bits.

References

References to the literature can be found in the following surveys:

- N. Schweikardt. Machine models and lower bounds for query processing. In Proc. PODS'07, pp. 41–52.
- N. Schweikardt. Machine models for query processing. SIGMOD Record 38(2), pp. 18–28, 2009.

Solutions to the exercises can be found in the following articles:

#1: S. Ganguly, A. Majumder: Deterministic K-set structure. Information Processing Letters 109(1), pp. 27–31, 2008.

#2: M. Grohe, A. Hernich, N. Schweikardt: Lower bounds for processing data with few random accesses to external memory. Journal of the ACM 56(3), 2009. — See Theorem 3.5.

#3: M. Henzinger, P. Raghavan, S. Rajagopalan: Computing on data streams. In *External Memory Algorithms*, J.M. Abello and J.S. Vitter (eds.). DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 50. AMS, New York, pp. 107–118, 1999. — See Theorem 6.

#4: G. Schnitger: Lecture notes on "Internet Algorithmen" (in German). Goethe-Universität Frankfurt am Main, 2009. `http://www.thi.informatik.uni-frankfurt.de/Internet0809/skript.pdf` — See Algorithm 4.20 on page 72.

Thank You!