

Machine Models and Lower Bounds for Query Processing

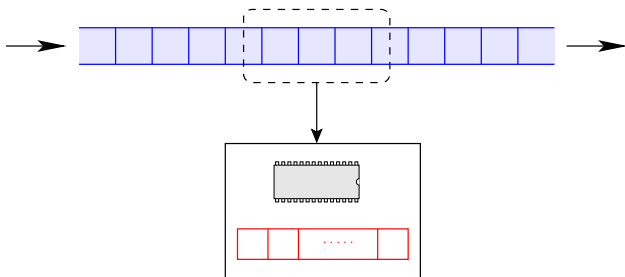
Nicole Schweikardt

Humboldt-University Berlin

PODS 2007

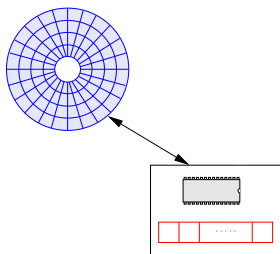
Beijing, China, 11 June 2007

Scenario 1: Data Streams



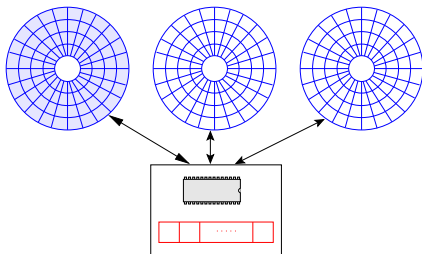
- **Data** are only read once.
- **Memory** is too small for storing all the data. At any point in time, only a small fraction of the data can be present in memory.

Scenario 2: Data in External Memory



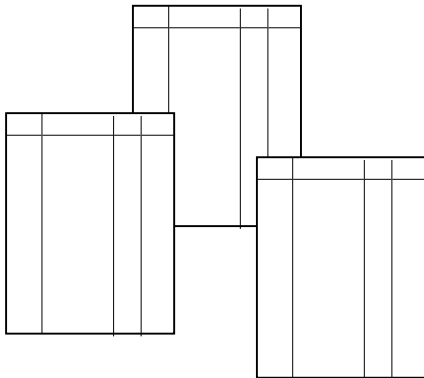
- Data in **external memory** (hard disk).
- **Internal memory** is too small for storing all the data.
- Sometimes, additional external memory devices can be used.

Scenario 2: Data in External Memory



- Data in **external memory** (hard disk).
- **Internal memory** is too small for storing all the data.
- Sometimes, additional external memory devices can be used.

Scenario 3: Data in a Relational Database



Classical Two-Pass Query Processing:

1. **Sort** the tables.
2. Evaluate relational algebra queries by **synchronized scans**.

Bottlenecks

- Internal memory is limited.
- Random access to data is problematic:
 - impossible for data streams.
 - expensive for data in external memory.
- But:
Sequentially streaming data through internal memory is relatively cheap.

Situation

- + efficient **streaming** or **external memory algorithms** for many concrete problems
- + database systems:
optimize the cost caused by external memory accesses
- + powerful tool for proving **lower bounds for data stream** problems:
communication complexity
- not clear, why certain problems do not (seem to) have efficient **external memory** algorithms
- classical complexity theory does not distinguish between
 - external memory and internal memory
 - random access to external memory and sequentially scanning external memory

Outline

Motivation

Data Streams

One External Memory Device

Several External Memory Devices

Finite Cursor Machines

Summary

Outline

Motivation

Data Streams

One External Memory Device

Several External Memory Devices

Finite Cursor Machines

Summary

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
~> process data “on-the-fly”
- cannot afford to store all the data ~> store a “sketch”
- data may arrive so rapidly that you cannot even afford to look at each incoming data item ~> “sampling”

For details see SIGMOD Tutorial by Graham Cormode and Minos Garofalakis

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

Naive Solution: 2 5 1 3 4 8 6 ... n requires n bits of storage

1	2	3	4	5	6	7	8	...	n

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

$$S := x_1 + x_2 + x_3 + x_4 + \dots + x_{n-1}$$

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
	✓								

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
	✓			✓					

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓			✓					

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓		✓					

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓					

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓			✓		

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓		

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

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

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

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

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

Missing Number Puzzle

MISSING NUMBER

Input: Stream $x_1, x_2, x_3, \dots, x_{n-1}$ of
 $n-1$ distinct numbers from $\{1, \dots, n\}$

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

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

1	2	3	4	5	6	7	8	\dots	n
✓	✓	✓	✓	✓	✓		✓	✓	✓

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

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

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

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

The MULTISET-EQUALITY Problem (1/3)

MULTISET-EQUALITY

Total input length: $N = O(m \cdot n)$ bits

Input: Two multisets $\{x_1, \dots, x_m\}$ and $\{y_1, \dots, y_m\}$ of bit-strings x_i, y_j
(for simplicity, all bit-strings have same length n)

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

Observation:

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

Proof:

- Use fact from **Communication Complexity:**

Communication Complexity

Yaos 2-Party Communication Model:

- 2 players: Alice & Bob
- both know a function $f : A \times B \rightarrow \{0, 1\}$
- 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

$$a = \{x_1, \dots, x_m\} \subseteq \{0, 1\}^n \quad \text{and} \quad b = \{y_1, \dots, y_m\} \subseteq \{0, 1\}^n$$

of n -bit-strings are equal, requires at least $\log \binom{2^n}{m}$ bits of communication.

Communication Complexity

Yaos 2-Party Communication Model:

- 2 players: Alice & Bob
- both know a function $f : A \times B \rightarrow \{0, 1\}$
- 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

$$a = \{x_1, \dots, x_m\} \subseteq \{0, 1\}^n \quad \text{and} \quad b = \{y_1, \dots, y_m\} \subseteq \{0, 1\}^n$$

of n -bit-strings are equal, requires at least $\log \binom{2^n}{m}$ bits of communication.

The MULTISET-EQUALITY Problem (1/3)

MULTISET-EQUALITY

Total input length: $N = O(m \cdot n)$ bits

Input: Two multisets $\{x_1, \dots, x_m\}$ and $\{y_1, \dots, y_m\}$ of bit-strings x_i, y_j
(for simplicity, all bit-strings have same length n)

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

Observation:

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

Proof:

- Use fact from **Communication Complexity**:

The MULTISSET-EQUALITY Problem (1/3)

MULTISSET-EQUALITY

Total input length: $N = O(m \cdot n)$ bits

Input: Two multisets $\{x_1, \dots, x_m\}$ and $\{y_1, \dots, y_m\}$ of bit-strings x_i, y_j
(for simplicity, all bit-strings have same length n)

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

Observation:

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

Proof:

- Use fact from **Communication Complexity**:

Deciding if two m -element sets of n -bit-strings are equal requires at least $\log \binom{2^n}{m}$ bits of communication.

- If $2^n = m^2$, then $\log \binom{2^n}{m} \geq m \cdot \log m$ bits of communication are necessary, and the total length of the corresponding MULTISSET-EQUALITY input is $N = \Theta(m \cdot \log m)$.

The MULTISSET-EQUALITY Problem (1/3)

MULTISSET-EQUALITY

Total input length: $N = O(m \cdot n)$ bits

Input: Two multisets $\{x_1, \dots, x_m\}$ and $\{y_1, \dots, y_m\}$ of bit-strings x_i, y_j
(for simplicity, all bit-strings have same length n)

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

Observation:

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

Proof:

- Use fact from **Communication Complexity**:
Deciding if two m -element sets of n -bit-strings are equal requires at least $\log \binom{2^n}{m}$ bits of communication.
- If $2^n = m^2$, then $\log \binom{2^n}{m} \geq m \cdot \log m$ bits of communication are necessary, and the total length of the corresponding MULTISSET-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 $\geq m \cdot \log m$ bits of communication are necessary for solving MULTISET-EQUALITY.
- A deterministic data stream algorithm solving MULTISET-EQUALITY with B bits of storage would lead to a communication protocol with B 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):

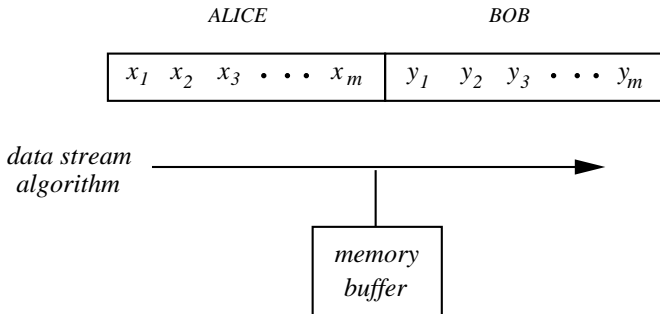
- Known: $N = \Theta(m \cdot \log m)$, and $\geq m \cdot \log m$ bits of communication are necessary for solving MULTISET-EQUALITY.
- A deterministic data stream algorithm solving MULTISET-EQUALITY with B bits of storage would lead to a communication protocol with B bits of communication.

- 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 $\geq m \cdot \log m$ bits of communication are necessary for solving MULTISET-EQUALITY.
- A deterministic data stream algorithm solving MULTISET-EQUALITY with B bits of storage would lead to a communication protocol with B bits of communication.

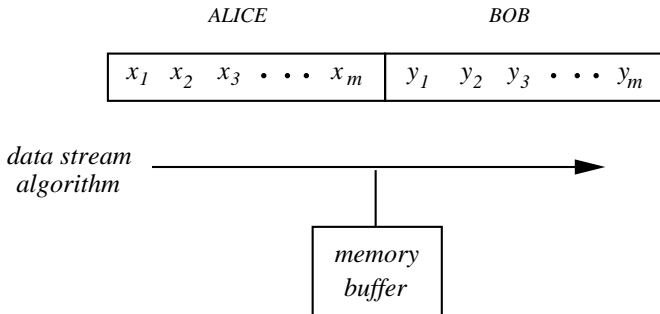


- Thus: Lower bound on communication complexity \rightsquigarrow lower bound on memory size of data stream algorithm

The MULTISET-EQUALITY Problem (2/3)

Proof (continued):

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

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 n -bit-strings $a_1, \dots, a_m, b_1, \dots, b_m$, the algorithm

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

Proof idea: Use “Fingerprinting”-techniques:

- represent $\{a_1, \dots, a_m\}$ by a polynomial $f(x) := \sum_{i=1}^m x^{a_i}$
- represent $\{b_1, \dots, b_m\}$ 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 $\{a_1, \dots, a_m\} = \{b_1, \dots, b_m\}$, then $f(x) = g(x)$, and thus the algorithm always accepts. If $\{a_1, \dots, a_m\} \neq \{b_1, \dots, b_m\}$, then there are at most $\text{degree}(f-g)$ many distinct r with $f(r) = g(r)$, and thus the algorithm rejects with high probability.

The MULTISSET-EQUALITY Problem (3/3)

Theorem:

The MULTISSET-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 n -bit-strings $a_1, \dots, a_m, b_1, \dots, b_m$, the algorithm

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

Proof idea: Use “Fingerprinting”-techniques:

- represent $\{a_1, \dots, a_m\}$ by a polynomial $f(x) := \sum_{i=1}^m x^{a_i}$
- represent $\{b_1, \dots, b_m\}$ 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 $\{a_1, \dots, a_m\} = \{b_1, \dots, b_m\}$, then $f(x) = g(x)$, and thus the algorithm always accepts. If $\{a_1, \dots, a_m\} \neq \{b_1, \dots, b_m\}$, then there are at most $\text{degree}(f-g)$ many distinct r with $f(r) = g(r)$, and thus the algorithm rejects with high probability.

The MULTISET-EQUALITY Problem (3/3)

Theorem:

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 n -bit-strings $a_1, \dots, a_m, b_1, \dots, b_m$, the algorithm

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

Proof idea: Use “Fingerprinting”-techniques:

- represent $\{a_1, \dots, a_m\}$ by a polynomial $f(x) := \sum_{i=1}^m x^{a_i}$
- represent $\{b_1, \dots, b_m\}$ 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 $\{a_1, \dots, a_m\} = \{b_1, \dots, b_m\}$, then $f(x) = g(x)$, and thus the algorithm always accepts. If $\{a_1, \dots, a_m\} \neq \{b_1, \dots, b_m\}$, then there are at most $\text{degree}(f-g)$ many distinct r with $f(r) = g(r)$, and thus the algorithm rejects with high probability.

The MULTISSET-EQUALITY Problem (3/3)

Theorem:

The MULTISSET-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 n -bit-strings $a_1, \dots, a_m, b_1, \dots, b_m$, the algorithm

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

Proof idea: Use “Fingerprinting”-techniques:

- represent $\{a_1, \dots, a_m\}$ by a polynomial $f(x) := \sum_{i=1}^m x^{a_i}$
- represent $\{b_1, \dots, b_m\}$ 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 $\{a_1, \dots, a_m\} = \{b_1, \dots, b_m\}$, then $f(x) = g(x)$, and thus the algorithm always accepts. If $\{a_1, \dots, a_m\} \neq \{b_1, \dots, b_m\}$, then there are at most $\text{degree}(f-g)$ many distinct r with $f(r) = g(r)$, and thus the algorithm rejects with high probability.

Outline

Motivation

Data Streams

One External Memory Device

Several External Memory Devices

Finite Cursor Machines

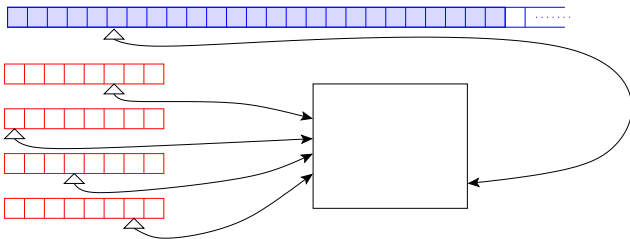
Summary

Goal: Machine Model for . . .

- fast & small **internal memory** **vs.** huge & slow **external memory**
 - external memory: **random access** **vs.** **sequential scans**
-

- ▶ **machine model** and complexity classes that **measure costs caused by external memory accesses**
- ▶ **lower bounds** for particular problems

Machine Model



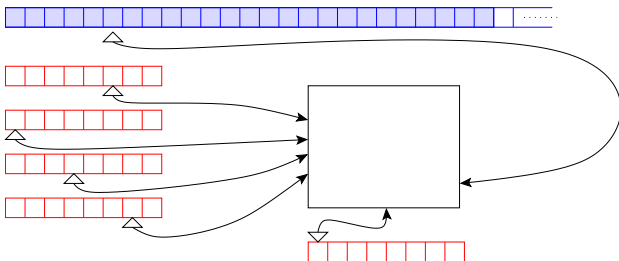
multi-tape Turing machine with

- one “long” tape (that represents external memory) limited access
- some “short” tapes (that represent internal memory) limited size

Input on the external memory tape.

If necessary: Output on the external memory tape.

Random Access



An additional **address tape** (as part of the internal memory)

- to specify addresses of tape positions on the external memory tape
- a particular state which allows to move the external memory tape's read/write head to the specified position in a single step

Head Reversals

- When the external memory tape models a hard disk or a data stream, it should be read only in **one** direction (from left to right).
- For our **lower bounds** we still allow head reversals on the external memory tape. (This makes our lower bound results only stronger.)
- **Allowing head reversals**, we can **ignore random access**, because each “random access jump” can be simulated by at most 2 head reversals.

Complexity Classes

Let $r : \mathbb{N} \rightarrow \mathbb{N}$ and $s : \mathbb{N} \rightarrow \mathbb{N}$.

A (r, s) -bounded TM is a Turing machine with

- one external memory tape,
- internal memory tapes of total length $s(N)$,
- less than $r(N)$ head reversals on the external memory tape

(where $N = \text{input length}$).

$ST(r, s) :=$ the class of all problems that can be solved by a deterministic (r, s) -bounded TM.

For classes R, S of functions we let

$ST(R, S) := \bigcup_{r \in R, s \in S} ST(r, s).$

Complexity Classes

Let $r : \mathbb{N} \rightarrow \mathbb{N}$ and $s : \mathbb{N} \rightarrow \mathbb{N}$.

A (r, s) -bounded TM is a Turing machine with

- one external memory tape,
- internal memory tapes of total length $s(N)$,
- less than $r(N)$ head reversals on the external memory tape

(where $N = \text{input length}$).

$ST(r, s) :=$ the class of all problems that can be solved by a deterministic (r, s) -bounded TM.

For classes R, S of functions we let

$$ST(R, S) := \bigcup_{r \in R, s \in S} ST(r, s).$$

Complexity Classes

Let $r : \mathbb{N} \rightarrow \mathbb{N}$ and $s : \mathbb{N} \rightarrow \mathbb{N}$.

A (r, s) -bounded TM is a Turing machine with

- one external memory tape,
- internal memory tapes of total length $s(N)$,
- less than $r(N)$ head reversals on the external memory tape

(where $N = \text{input length}$).

$ST(r, s) :=$ the class of all problems that can be solved by a deterministic (r, s) -bounded TM.

For classes R, S of functions we let

$$ST(R, S) := \bigcup_{r \in R, s \in S} ST(r, s).$$

Complexity Classes

ST(1, s):

- input is a **data stream**,
- only **internal** memory available for the computation.

ST(r, s):

- input on the hard disk,
- this hard disk may be used throughout the computation,
- $\leq r(N)$ sequential scans of the hard disk,
- internal memory of size $\leq s(N)$.

Complexity Classes

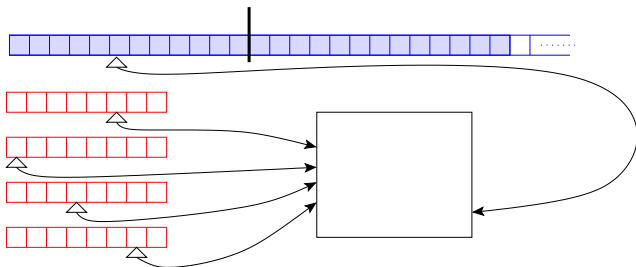
ST(1, s):

- input is a **data stream**,
- only **internal** memory available for the computation.

ST(r, s):

- input on the hard disk,
- this hard disk may be used throughout the computation,
- $\leq r(N)$ sequential scans of the hard disk,
- internal memory of size $\leq s(N)$.

An Easy Observation



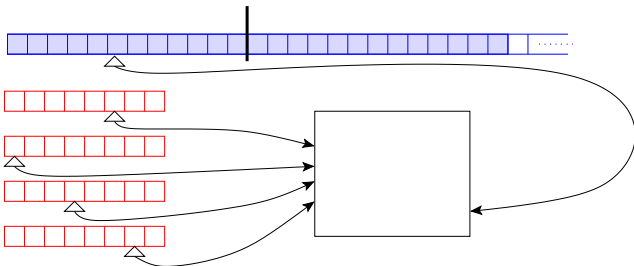
Fact:

During an (r, s) -bounded computation, **only $O(r(N) \cdot s(N))$ bits can be communicated** between the first and the second half of the external memory tape.

Consequence:

Lower bounds on **communication complexity** lead to lower bounds for the $ST(\dots)$ classes.

An Easy Observation



Fact:

During an (r, s) -bounded computation, **only $O(r(N) \cdot s(N))$ bits can be communicated** between the first and the second half of the external memory tape.

Consequence:

Lower bounds on **communication complexity** lead to lower bounds for the $ST(\dots)$ classes.

Some Results

A lower bound for Sorting:

SORTING

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

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

Output: x_1, \dots, x_m sorted in ascending order

Theorem:

(Grohe, Koch, S., ICALP'05)

For all $r, s : \mathbb{N} \rightarrow \mathbb{N}$ we have: **SORTING** \in **ST**(r, s) $\iff r(N) \cdot s(N) \in \Omega(N)$.

A Hierarchy of Head Reversals:

Theorem:

(Hernich, S., 2006)

For every logspace-computable function r with $r(N) \in o\left(\frac{N}{\log^2 N}\right)$, and

for every class S of functions such that $O(\log N) \subseteq S \subseteq o\left(\frac{N}{r(N) \cdot \log N}\right)$ we have:

ST($r(N), S$) \subsetneq **ST**($r(N)+1, S$)

Remark: An analogous result also holds for **randomised** versions of **ST**(\cdot, \cdot)

Some Results

A lower bound for Sorting:

SORTING

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

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

Output: x_1, \dots, x_m sorted in ascending order

Theorem:

(Grohe, Koch, S., ICALP'05)

For all $r, s : \mathbb{N} \rightarrow \mathbb{N}$ we have: $\text{SORTING} \in \text{ST}(r, s) \iff r(N) \cdot s(N) \in \Omega(N)$.

A Hierarchy of Head Reversals:

Theorem:

(Hernich, S., 2006)

For every logspace-computable function r with $r(N) \in o\left(\frac{N}{\log^2 N}\right)$, and

for every class S of functions such that $O(\log N) \subseteq S \subseteq o\left(\frac{N}{r(N) \cdot \log N}\right)$ we have:

$\text{ST}(r(N), S) \subsetneq \text{ST}(r(N)+1, S)$

Remark: An analogous result also holds for **randomised** versions of $\text{ST}(\cdot, \cdot)$

XPath Query Processing on XML Streams

XML Stream

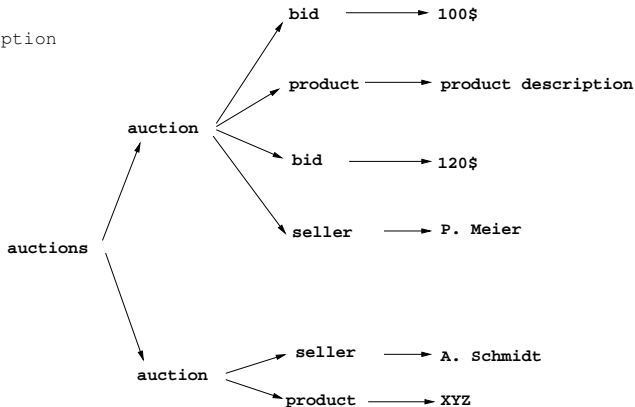
```

<auctions>
  <auction>
    <bid>
      100$
    </bid>
    <product>
      product description
    </product>
    <bid>
      120$
    </bid>
    <seller>
      P. Meier
    </seller>
  </auction>
  <auction>
    <seller>
      A. Schmidt
    </seller>
    <product>
      XYZ
    </product>
  </auction>
</auctions>

```

Example: `//auction[seller='P. Meier']/bid`

XML Tree



XPath Query Processing on XML Streams

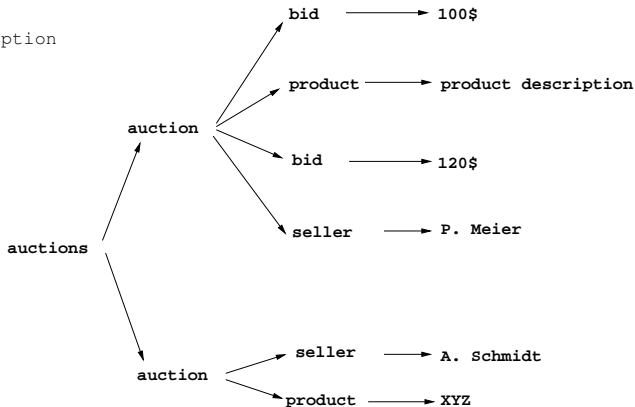
XML Stream

```

<auctions>
  <auction>
    <bid>
      100$
    </bid>
    <product>
      product description
    </product>
    <bid>
      120$
    </bid>
    <seller>
      P. Meier
    </seller>
  </auction>
  <auction>
    <seller>
      A. Schmidt
    </seller>
    <product>
      XYZ
    </product>
  </auction>
</auctions>
  
```

Example: `// auction[seller='P. Meier']/bid`

XML Tree



XPath Query Processing on XML Streams

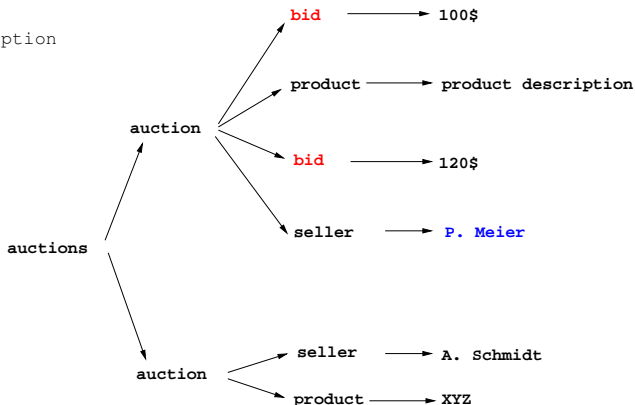
XML Stream

```

<auctions>
  <auction>
    <bid>
      100$
    </bid>
    <product>
      product description
    </product>
    <bid>
      120$
    </bid>
    <seller>
      P. Meier
    </seller>
  </auction>
  <auction>
    <seller>
      A. Schmidt
    </seller>
    <product>
      XYZ
    </product>
  </auction>
</auctions>
  
```

Example: `//auction[seller='P. Meier']/bid`

XML Tree



XPath Query Processing on XML Streams

- **XPath:**

a node-selecting XML query language, standardised by the W3C, the “navigation component” of XQuery and XSLT

- **Core XPath** (*Gottlob, Koch, 2000*):

A logically “clean” fragment of XPath.

Expressive power of Core XPath: weaker than **node-selecting** formulas from Monadic Second-Order Logic (**MSO**)

Q-EVALUATION (for a Core XPath query Q)

Input: XML-document D

Task: Compute the set of nodes selected by Q in S .

Q-FILTERING (for a Core XPath query Q)

Input: XML-document D

Question: Does the query Q select at least one node in D ?

XPath Query Processing on XML Streams

- XPath:

a node-selecting XML query language, standardised by the W3C, the “navigation component” of XQuery and XSLT

- Core XPath (*Gottlob, Koch, 2000*):

A logically “clean” fragment of XPath.

Expressive power of Core XPath: weaker than node-selecting formulas from Monadic Second-Order Logic (MSO)

Q-EVALUATION (for a Core XPath query Q)

Input: XML-document D

Task: Compute the set of nodes selected by Q in S .

Q-FILTERING (for a Core XPath query Q)

Input: XML-document D

Question: Does the query Q select at least one node in D ?

XPath Evaluation / Filtering on XML Streams

Upper bounds (algorithms, systems):

- large number of clever contributions by several research groups
- various XPath fragments considered
- many approaches based on finite automata, pushdown automata, or networks of automata

Lower bounds (on memory for XPath processing on XML streams):

- work by Bar-Yossef, Fontoura, Josifovski (PODS'04 and PODS'05)
 - ▶ introduce particular fragments of XPath
 - ▶ PODS'04: lower bounds for XPath **filtering** on XML streams
 - ▶ PODS'05: lower bounds for XPath **evaluation** on XML streams:
 - ▶ Proof method: **communication complexity**

XPath Evaluation / Filtering on XML Streams

Upper bounds (algorithms, systems):

- large number of clever contributions by several research groups
- various XPath fragments considered
- many approaches based on finite automata, pushdown automata, or networks of automata

Lower bounds (on memory for XPath processing on XML streams):

- work by Bar-Yossef, Fontoura, Josifovski (PODS'04 and PODS'05)
 - ▶ introduce particular fragments of XPath
 - ▶ PODS'04: lower bounds for XPath **filtering** on XML streams
 - ▶ PODS'05: lower bounds for XPath **evaluation** on XML streams:
 - ▶ Proof method: **communication complexity**

XPath Processing on XML Stored in External Memory

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

- (a) For every Core XPath query Q we have: $Q\text{-FILTERING} \in ST(1, O(\text{height}(D)))$
 and $Q\text{-EVALUATION} \in ST(2, O(\text{height}(D) + \log(\text{size}(D))))$
- (b) There is a Core XPath query Q such that for all r, s with $r(D) \cdot s(D) \in o(\text{height}(D))$ we have: $Q\text{-FILTERING} \notin ST(r, s)$.

Proof idea:

(b): **Communication complexity** leads to a lower bound for the amount of information that has to be transported over the middle of the document ...

Consider the **DISJOINT-SETS** problem:

Input: Two sets $S_1, S_2 \subseteq \{1, \dots, n\}$.

Question: Is $S_1 \cap S_2 = \emptyset$?

Known: Requires at least n bits of communication.

XPath Processing on XML Stored in External Memory

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

- (a) For every Core XPath query Q we have: $Q\text{-FILTERING} \in \text{ST}(1, O(\text{height}(D)))$
 and $Q\text{-EVALUATION} \in \text{ST}(2, O(\text{height}(D) + \log(\text{size}(D))))$
- (b) There is a Core XPath query Q such that for all r, s with $r(D) \cdot s(D) \in o(\text{height}(D))$ we have: $Q\text{-FILTERING} \notin \text{ST}(r, s)$.

Proof idea:

(b): **Communication complexity** leads to a lower bound for the amount of information that has to be transported over the middle of the document ...

Consider the **DISJOINT-SETS** problem:

Input: Two sets $S_1, S_2 \subseteq \{1, \dots, n\}$.

Question: Is $S_1 \cap S_2 = \emptyset$?

Known: Requires at least n bits of communication.

XPath Processing on XML Stored in External Memory

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

- (a) For every Core XPath query Q we have: $Q\text{-FILTERING} \in \text{ST}(1, O(\text{height}(D)))$
 and $Q\text{-EVALUATION} \in \text{ST}(2, O(\text{height}(D) + \log(\text{size}(D))))$
- (b) There is a Core XPath query Q such that for all r, s with $r(D) \cdot s(D) \in o(\text{height}(D))$ we have: $Q\text{-FILTERING} \notin \text{ST}(r, s)$.

Proof idea:

(b): **Communication complexity** leads to a lower bound for the amount of information that has to be transported over the middle of the document ...

Consider the **DISJOINT-SETS** problem:

Input: Two sets $S_1, S_2 \subseteq \{1, \dots, n\}$.

Question: Is $S_1 \cap S_2 = \emptyset$?

Known: Requires at least n bits of communication.

XPath Processing on XML Stored in External Memory

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

- (a) For every Core XPath query Q we have: $Q\text{-FILTERING} \in ST(1, O(\text{height}(D)))$
 and $Q\text{-EVALUATION} \in ST(2, O(\text{height}(D) + \log(\text{size}(D))))$
- (b) There is a Core XPath query Q such that for all r, s with $r(D) \cdot s(D) \in o(\text{height}(D))$ we have: $Q\text{-FILTERING} \notin ST(r, s)$.

Proof idea:

(b): **Communication complexity** leads to a lower bound for the amount of information that has to be transported over the middle of the document ...

Consider the **DISJOINT-SETS** problem:

Input: Two sets $S_1, S_2 \subseteq \{1, \dots, n\}$.

Question: Is $S_1 \cap S_2 = \emptyset$?

Known: Requires at least n bits of communication.

XPath Processing on XML Stored in External Memory

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

- (a) For every Core XPath query Q we have: $Q\text{-FILTERING} \in ST(1, O(\text{height}(D)))$
 and $Q\text{-EVALUATION} \in ST(2, O(\text{height}(D) + \log(\text{size}(D))))$
- (b) There is a Core XPath query Q such that for all r, s with $r(D) \cdot s(D) \in o(\text{height}(D))$ we have: $Q\text{-FILTERING} \notin ST(r, s)$.

Proof idea:

(b): **Communication complexity** leads to a lower bound for the amount of information that has to be transported over the middle of the document ...

Consider the **DISJOINT-SETS** problem:

Input: Two sets $S_1, S_2 \subseteq \{1, \dots, n\}$.

Question: Is $S_1 \cap S_2 = \emptyset$?

Known: Requires at least n bits of communication.

... Proof of (b), continued

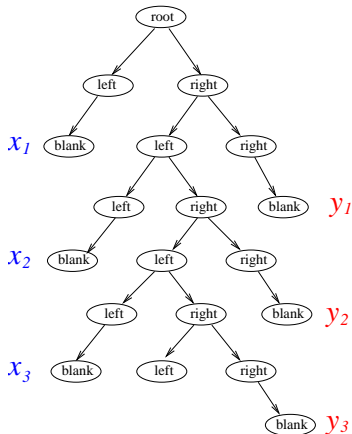
- Encode the DISJOINT-SETS problem by XML trees:

- $S_1, S_2 \subseteq \{1, \dots, n\}$ are encoded via

$$x_i = 1 \iff i \in S_1,$$

$$y_i = 1 \iff i \in S_2.$$

- $n \approx$ height of document tree = amount of information that must be transported over the middle of the document.



- Core XPath formulation of the DISJOINT-SETS problem:

```
//*[right/right/1]/left/1
```

... Proof of (b), continued

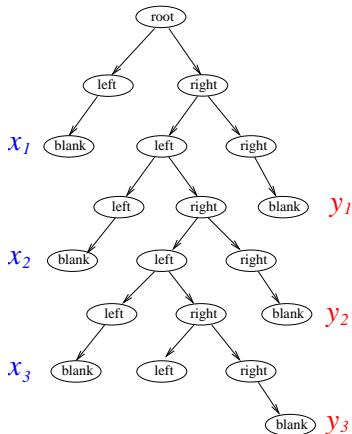
- Encode the DISJOINT-SETS problem by XML trees:

- $S_1, S_2 \subseteq \{1, \dots, n\}$ are encoded via

$$x_i = 1 \iff i \in S_1,$$

$$y_i = 1 \iff i \in S_2.$$

- $n \approx$ height of document tree = amount of information that must be transported over the middle of the document.



- Core XPath formulation of the DISJOINT-SETS problem:

```
//*[right/right/1]/left/1
```

... Proof of (b), continued

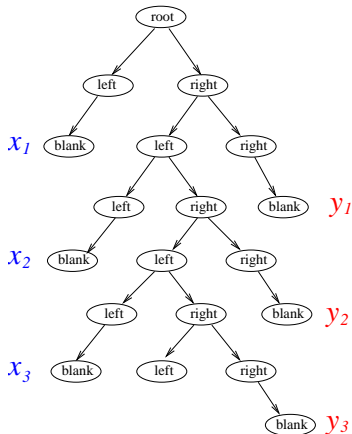
- Encode the DISJOINT-SETS problem by XML trees:

- $S_1, S_2 \subseteq \{1, \dots, n\}$ are encoded via

$$x_i = 1 \iff i \in S_1,$$

$$y_i = 1 \iff i \in S_2.$$

- $n \approx$ height of document tree = amount of information that must be transported over the middle of the document.



- Core XPath formulation of the DISJOINT-SETS problem:

```
//*[right/right/1]/left/1
```

XPath Processing on XML Stored in External Memory

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

- (a) For every Core XPath query Q we have: $Q\text{-FILTERING} \in ST(1, O(\text{height}(D)))$
and $Q\text{-EVALUATION} \in ST(2, O(\text{height}(D) + \log(\text{size}(D))))$
- (b) There is a Core XPath query Q such that for all r, s with $r(D) \cdot s(D) \in o(\text{height}(D))$ we have: $Q\text{-FILTERING} \notin ST(r, s)$.

Proof idea: (a)

Q-FILTERING Problem: For every Core XPath query Q there is a bottom-up tree automaton that solves the filtering problem for Q .

A run of this automaton can be simulated during a single forward-scan of the XML document. \rightsquigarrow solution of the Q-FILTERING problem

For the **Q-EVALUATION problem** use **selecting tree automata**:

- (1) forward scan of the XML document: simulate the run of a bottom-up tree automaton, use external memory to decorate the "closing bracket" of each node with the automaton's state at that node.
- (2) backward scan of the "decorated" XML document: simulate the run of a top-down tree automaton, output the indices of those nodes at which a special **selecting state** is assumed.

XPath Processing on XML Stored in External Memory

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

- (a) For every Core XPath query Q we have: $Q\text{-FILTERING} \in ST(1, O(\text{height}(D)))$
and $Q\text{-EVALUATION} \in ST(2, O(\text{height}(D) + \log(\text{size}(D))))$
- (b) There is a Core XPath query Q such that for all r, s with $r(D) \cdot s(D) \in o(\text{height}(D))$ we have: $Q\text{-FILTERING} \notin ST(r, s)$.

Proof idea: (a)

Q-FILTERING Problem: For every Core XPath query Q there is a **bottom-up tree automaton** that solves the filtering problem for Q .

A run of this automaton can be simulated during a single forward-scan of the XML document. \rightsquigarrow solution of the $Q\text{-FILTERING}$ problem

For the $Q\text{-EVALUATION}$ problem use **selecting tree automata**:

- (1) **forward scan of the XML document:** simulate the run of a bottom-up tree automaton, use external memory to decorate the "closing bracket" of each node with the automaton's state at that node.
- (2) **backward scan of the "decorated" XML document:** simulate the run of a top-down tree automaton, output the indices of those nodes at which a special **selecting state** is assumed.

XPath Processing on XML Stored in External Memory

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

- (a) For every Core XPath query Q we have: $Q\text{-FILTERING} \in \text{ST}(1, O(\text{height}(D)))$
 and $Q\text{-EVALUATION} \in \text{ST}(2, O(\text{height}(D) + \log(\text{size}(D))))$
- (b) There is a Core XPath query Q such that for all r, s with $r(D) \cdot s(D) \in o(\text{height}(D))$ we have: $Q\text{-FILTERING} \notin \text{ST}(r, s)$.

Proof idea: (a)

Q-FILTERING Problem: For every Core XPath query Q there is a **bottom-up tree automaton** that solves the filtering problem for Q .

A run of this automaton can be simulated during a single forward-scan of the XML document. \rightsquigarrow solution of the Q-FILTERING problem

For the **Q-EVALUATION problem** use **selecting tree automata**:

- forward scan of the XML document:** simulate the run of a bottom-up tree automaton, use external memory to decorate the “closing bracket” of each node with the automaton’s state at that node.
- backward scan of the “decorated” XML document:** simulate the run of a top-down tree automaton, output the indices of those nodes at which a special **selecting state** is assumed.

An Open Question:

We have just seen that for every Core XPath query Q :

$$Q\text{-EVALUATION} \in ST(2, O(\text{height}(D) + \log(\text{size}(D))))$$

by an algorithm which performs **one forward scan** and **one backward scan**, and which needs to **write onto the external memory** tape during the forward scan.

Open questions:

- ▶ Is a **backward scan** really necessary here?

Obvious: a single forward scan doesn't suffice. But what about 2 forward scans?

- ▶ Is **writing** to the external memory tape really necessary here?

Outline

Motivation

Data Streams

One External Memory Device

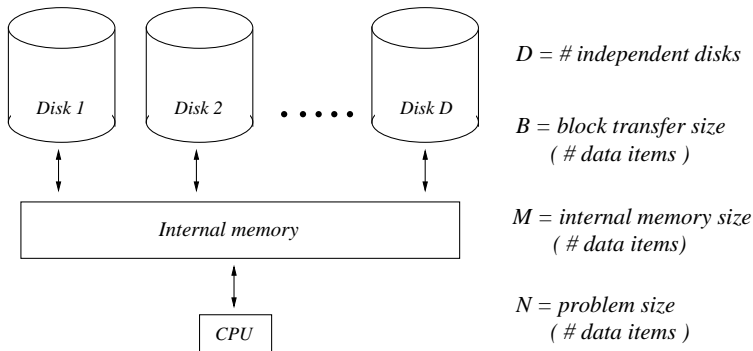
Several External Memory Devices

Finite Cursor Machines

Summary

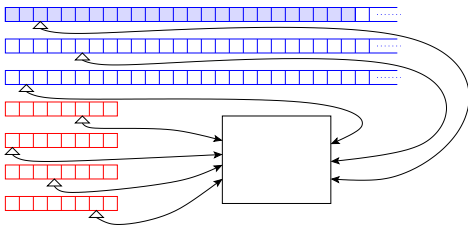
The Parallel Disk Model (PDM)

Introduced by Vitter and Shriver, 1994



- + good for designing and analysing external memory algorithms
- no distinction between **streaming** and **random access**
- not so suitable for proving lower bounds

Turing Machine Model



multi-tape Turing machine with

- t “long” tapes (that represent t external memory devices) limited access
- some “short” tapes (that represent internal memory) limited size

Input on the first external memory tape.

If necessary: Output on the t -th external memory tape.

$ST(r, s, t)$: complexity class similar to $ST(r, s)$, but with t long tapes

$ST(R, S, O(1)) := \bigcup_{t \geq 1} ST(R, S, t)$

The Sorting-Problem

SORTING

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

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

Output: x_1, \dots, x_m sorted in ascending order

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

Theorem:

(Chen, Yap, 1991)

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

Proof method: Refinement of Merge-Sort.

Question: Is this optimal? I.e.: What about $o(\log n)$ head reversals?

Lower Bound for Sorting with ≥ 2 EM-tapes

Problem:

An additional external memory tape can be used to move around large parts of the input (with just 2 head reversals).

↪ 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 $t \geq 2$ external memory tapes, 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 ≥ 2 EM-tapes

Problem:

An additional external memory tape can be used to move around large parts of the input (with just 2 head reversals).

↪ 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 $t \geq 2$ external memory tapes, 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 ≥ 2 EM-Tapes

Problem:

Turing machines 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 tape.

Lower Bound for Sorting

Theorem:

$\text{SORTING} \notin \text{ST}(o(\log N), N^{1-\varepsilon}, O(1))$

(Grohe, S., PODS'05)

(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. Simulate (r, s, t) -bounded TMs by list machines.

3. Prove that list machines cannot sort (\dots use combinatorics).

Randomised ST-Classes: RST and co-RST

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

An **RST**-machine 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**-machine 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)

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

Randomised ST-Classes: RST and co-RST

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

An **RST**-machine 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**-machine 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)

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

Randomised ST-Classes: RST and co-RST

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

An **RST**-machine 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**-machine 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)

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

Consequences

- Separation of deterministic, randomised, and nondeterministic $ST(\dots)$ -classes:

$NST(R, S, O(1))$

|
 $RST(R, S, O(1))$

|
 $ST(R, S, O(1))$

← $MULTISET-EQUALITY \in NST(3, O(\log N), 2)$

← $MULTISET-EQUALITY \in \text{co-RST}(2, O(\log N), 1)$

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

- Lower bound for the worst-case data complexity of the evaluation of XPath queries against XML-streams:

Theorem: There is an XPath query Q such that

$Q\text{-FILTERING} \notin \text{co-RST}(o(\log N), N^{1-\epsilon}, O(1)).$

Consequences

- Separation of deterministic, randomised, and nondeterministic $ST(\dots)$ -classes:

$NST(R, S, O(1))$

|
 $RST(R, S, O(1))$

|
 $ST(R, S, O(1))$

← $MULTISET-EQUALITY \in NST(3, O(\log N), 2)$

← $MULTISET-EQUALITY \in \text{co-RST}(2, O(\log N), 1)$

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

- Lower bound for the worst-case data complexity of the evaluation of [XPath](#) queries against XML-streams:

Theorem: There is an XPath query Q such that

$Q\text{-FILTERING} \notin \text{co-RST}(o(\log N), N^{1-\epsilon}, O(1)).$

Consequences

- Separation of deterministic, randomised, and nondeterministic $ST(\dots)$ -classes:

$NST(R, S, O(1))$

|
 $RST(R, S, O(1))$

|
 $ST(R, S, O(1))$

← $MULTISET-EQUALITY \in NST(3, O(\log N), 2)$

← $MULTISET-EQUALITY \in \text{co-RST}(2, O(\log N), 1)$

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

- Lower bound for the worst-case data complexity of the evaluation of [XPath](#) queries against XML-streams:

Theorem: There is an [XPath](#) query Q such that

$Q\text{-FILTERING} \notin \text{co-RST}(o(\log N), N^{1-\epsilon}, O(1)).$

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 $\left(o\left(\frac{\log N}{\log \log N}\right), N^{1-\epsilon}, O(1) \right)$ (for every $\epsilon > 0$)

Note:

All currently known lower bound proofs for (deterministic or randomized) ST-classes with ≥ 2 em-tapes rely on

- (1) a **key lemma** which **reduces** the problem of proving lower bounds for ST-machines **to a purely combinatorial problem**
(see Lemma 4.13 in the PODS'07 proceedings)
- (2) a clever use of combinatorics

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 $\varepsilon > 0$)

Note:

All currently known lower bound proofs for (deterministic or randomized) ST-classes with ≥ 2 em-tapes rely on

- (1) a **key lemma** which **reduces** the problem of proving lower bounds for ST-machines **to a purely combinatorial problem**
(see Lemma 4.13 in the PODS'07 proceedings)
- (2) a clever use of combinatorics

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 $\varepsilon > 0$)

Note:

All currently known lower bound proofs for (deterministic or randomized) ST-classes with ≥ 2 em-tapes rely on

- (1) a **key lemma** which **reduces** the problem of proving lower bounds for ST-machines **to a purely combinatorial problem**
(see Lemma 4.13 in the PODS'07 proceedings)
- (2) a clever use of combinatorics

Some Future Tasks

- (1) All currently known lower bounds for the ST-models with ≥ 2 em-tapes consider **only $o(\log N)$ head reversals**.

To do:

Show lower bounds for appropriate problems in a setting where **$\Omega(\log N)$ head reversals** and several em-tapes are available.

Caveat: It is known that $\text{LOGSPACE} \subseteq \text{ST}(O(\log N), O(1), 2)$.

- (2) Study the **related model** with **several em-tapes** and **intermediate sorting steps**.

This model is known as the **StrSort** model.

(Aggarwal, Datar, Rajagopalan, Ruhl, FOCS'04 & Ruhl's PhD thesis, 2003)

Some Future Tasks

- (1) All currently known lower bounds for the ST-models with ≥ 2 em-tapes consider only $o(\log N)$ head reversals.

To do:

Show lower bounds for appropriate problems in a setting where $\Omega(\log N)$ head reversals and several em-tapes are available.

Caveat: It is known that $\text{LOGSPACE} \subseteq \text{ST}(O(\log N), O(1), 2)$.

- (2) Study the related model with several em-tapes and intermediate sorting steps.

This model is known as the StrSort model.

(Aggarwal, Datar, Rajagopalan, Ruhl, FOCS'04 & Ruhl's PhD thesis, 2003)

Some Future Tasks

- (1) All currently known lower bounds for the ST-models with ≥ 2 em-tapes consider only $o(\log N)$ head reversals.

To do:

Show lower bounds for appropriate problems in a setting where $\Omega(\log N)$ head reversals and several em-tapes are available.

Caveat: It is known that $\text{LOGSPACE} \subseteq \text{ST}(O(\log N), O(1), 2)$.

- (2) Study the related model with several em-tapes and intermediate sorting steps.

This model is known as the StrSort model.

(Aggarwal, Datar, Rajagopalan, Ruhl, FOCS'04 & Ruhl's PhD thesis, 2003)

Outline

Motivation

Data Streams

One External Memory Device

Several External Memory Devices

Finite Cursor Machines

Summary

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** (instead of Turing 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** (instead of Turing 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

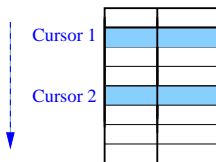
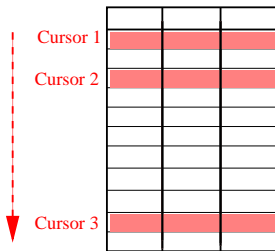
Finite Cursor Machines

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** (instead of Turing 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



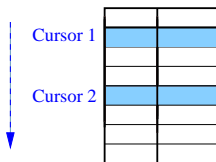
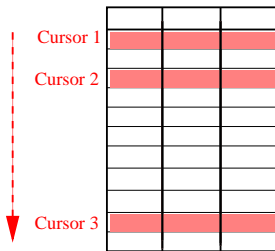
Finite Cursor Machines

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** (instead of Turing 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



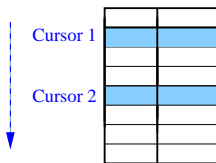
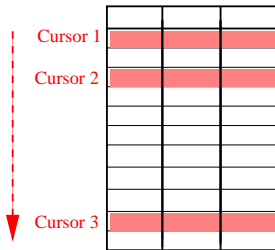
Finite Cursor Machines

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** (instead of Turing 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

Is $R \bowtie_{x_1=y_1} (S \bowtie_{x_2=y_1} T)$ 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 $n \cdot \log n$

Outline

Motivation

Data Streams

One External Memory Device

Several External Memory Devices

Finite Cursor Machines

Summary

Summary

- **Finite Cursor Machines:** an abstract model for database query processing
- **Communication Complexity** \rightsquigarrow
tight lower bounds in the data stream scenario and in the scenario with only **one single external memory device**
- Additional external memory devices render this approach useless
- Still, lower bound proofs exist also for this scenario . . . even for **randomised** computations.
- **Application:** Lower bounds for the worst case data complexity of query evaluation for **XPath, XQuery, and relational algebra**

Future Tasks

- (1) **FCMs:** Is there a **Boolean query from Relational Algebra** that **cannot** be computed by any **composition of FCMs and sorting** operations?
- (2) **ST-model with several em-tapes:** Show lower bounds for appropriate problems in a setting where $\Omega(\log N)$ **head reversals** and several em-tapes are available.
Caveat: It is known that $\text{LOGSPACE} \subseteq \text{ST}(O(\log N), O(1), 2)$.
- (3) **ST-model with one em-tape:**
Are backward scans really necessary for Core XPath query evaluation?
- (4) **More general models:**
Study the extension of the ST-model with intermediate sorting steps.
- (5) **The Parallel Disk Model:** Show lower bound for the sorting problem without using the indivisibility assumption. (According lower bounds *with* the indivisibility assumption are known, see work by Aggarwal and Vitter.)
- (6) **Complexity Theory:** Can the sorting problem be solved by a linear time multi-tape Turing machine?