6.897: Algorithms for Massive Data Sets

Piotr Indyk
indyk@theory.lcs.mit.edu

MIT

Contact

- Lecturer (secretary, webmaster): Piotr Indyk
- Time/place: Tue/Thu, 4 pm, 34-302
- Web: `theory.lcs.mit.edu/~indyk`
- E-mail: `indyk@theory.lcs.mit.edu`

Course structure

- Lectures
  - Lectures 1 - 15, 28: myself
  - Lectures 16 - 27: student presentations
- Grading
  - Two homeworks: early Oct, mid Nov
  - Presentation instead of 1 homework
    - Roughly 1 paper per presentation
    - Two people per slot OK
    - Before presentation: 2 meetings
    - Assignment on first-come-first-serve basis, so send me an e-mail ASAP (and specify several options)

Course content

Algorithmic techniques for dealing with massive data sets.

Constraints:

- Fairly general applicability
- Of theoretical interest
- Used in practice

Warnings

- "Theory" course ahead !
  - a healthy dose of math
  - knowledge of 6.046-level material required
  - no programming

"How to fit a massive data set into a 10MB Athena account ?"

- Course under construction !
  - shifting topics
  - suggestions welcome

Massive data sets

Examples:

- The Web: > 2 billion pages, each $\approx 1KB$
- Human genome: ≈ 3 billion base pairs
- WalMart market-basket data: 24 TB
- Sloan Digital Sky Survey: 40 TB
- AT&T call graphs
- etc.

Goals

- Would like to:
  - Compute statistics (mean, median, variance, etc.)
  - Find interesting patterns (clustering, SVD)
  - Answer queries (find an object similar to a given one)
- Also, need to:
  - Compress
  - Correct errors and erasures
- Problems considered before, many algorithms...
- ...but the scale of the problems unprecedented
- E.g., consider clustering:
  - Hundreds of algorithms known
  - Many popular ones have quadratic running time
  - Using them on gigabytes of data infeasible

Constraints

- Time
- Space
- Computational model:
  - Main memory: the "6.046 model"
  - External memory (disk)
  - Streaming: the data comes in a stream

Definition by example: the distinct elements problem

- Problem: given a sequence $A = a_1, \ldots a_n$, where $a_i \in \{1 \ldots m\}$, compute the number of *distinct* elements in A (denoted by $|A|$)
- Natural and popular statistics, e.g.:
  - Given the list of transactions, compute the number of different customers (i.e., credit card numbers)
  - What is the size of the web vocabulary ?
- Will see how different model require different solutions

Distinct elements: main memory

Algorithm H:

- Initialize $C = 0$, an array $B[1 \ldots O(n)]$ and a hash function $h : \{1 \ldots m\} \rightarrow \{1 \ldots O(n)\}$
- For each a_i:
  - Compute $j = h(a_i)$
  - Check if a_i occurs in the list pointed to by $B[j]$
  - If not, $C := C + 1$ and add a_i to the list

Assuming we have enough memory, and that $h(\cdot)$ is "random enough", the running time is $O(n)$.

External memory

External memory model:

- M units of main memory
- Input size n, $n >> M$
- Data is stored on *disk:*
  - Space divided into blocks, each of size $B \leq M$
  - Transfering one block of data into the main memory takes unit time
- Main memory operations for free

How realistic is this model ?

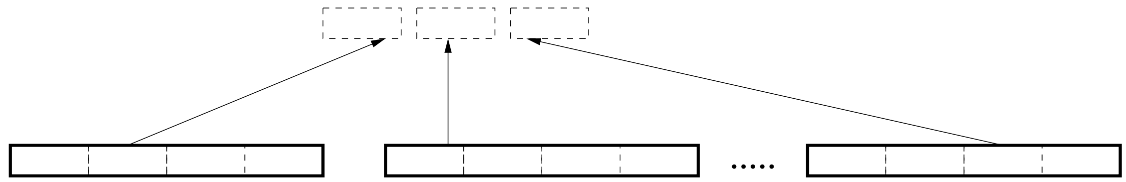
- Free operations in main memory:
  - Processors' speed: a few GHz
  - Seek time: ≈ 0.01 s
- Block transfer:
  - Transfer rate (say) 100 MB/s
  - If we set (say) $B = 100KB$, then data transfer essentially for free
  - Makes sense to just count the number of seeks

Distinct elements in external memory

- Algorithm H performs $\Theta(n)$ block accesses
- This is *really* bad (e.g., take $n = 10^9$)
- Need a different approach

Sorting in external memory

- Merge sort:
  - Split the data into M/B segments
  - Recursively sort each segment
  - Merge the segments using n/B block accesses:



- Sorting time: $(n/B)\log_{M/B} n$.

- After sorting the data, can compute $|A|$ in one pass:

$$1, 1, 6, 7, 9, 9, 9, 9, 40, 145, 145, 145, \ldots$$

- Total cost: $((n/B)+1)\log_{M/B}$

Streaming model

What if the data resides on CD-ROM, or comes straight from the cash-register ?

Streaming model:

- M units of main memory, $M << n$.
- The data comes in a stream. E.g., at any step we can request the next a_i
- Assume we know n (can get rid of this assumption)

Previous algorithms not applicable.

Bad news

Theorem: In order to solve the distinct elements problem, any deterministic* algorithm needs $\Omega(n)$ bits of space.

Sketch of the proof:

- By contradiction, assume there is a *deterministic* algorithm F using $M = o(n)$ bits of space

- Take any $S \subset \{1 \ldots m\}$; $n \leq m$.

- Feed S to F; denote the state of A at that point by X

- We can now check if $i \in S$ for any $i \in \{1 \ldots M\}$
  - Feed F with i
  - The number of distinct elements increased iff $i \notin S$

- This means we can recover S from X
- Therefore, the number of different X's must be at least 2^m

Good news

- The lower bound does not exclude the option of computing the value *approximately* !

- Will show how to construct an algorithm which for a *fixed* t, outputs either YES or NO, so that:
  - if $|A| \leq t$, the algorithm will say NO with probability $> 1/4 = 0.25$
  - if $|A| > 2t$, the algorithm will say NO with probability $< 1/e^2 = 0.136...$

- By running $O(1/\delta)$ algorithms in parallel, we can amplify the probability of error to any $\delta > 0$; also, can approximate up to $1 + \epsilon$ for any $\epsilon > 0$.

- On top of that, by running $\log n$ algorithms in parallel for $t = 2, 4, 8, \ldots n$, we can estimate $|A|$ up to a factor of $2(1 + \epsilon)$

- Total space used: $O(\log n/\epsilon^2 \cdot \log(1/\delta))$ bits.

The algorithm

- Initialize a hash function $h : \{1 \dots m\} \to \{1 \dots t\}$
- Initialize the answer to NO
- For each a_i:
  - if $h(a_i) = 1$, then set the answer to YES

Note: the algorithm uses only 1 bit of storage !
(not counting the random bits, will return to it later)

Correctness

What is the probability that after reading the stream, the algorithm will report NO ?

- Let $\tilde{A}$ be the *set* of distinct elements in A
- The algorithm reports NO iff $h^{-1}(1) \cap \tilde{A} = \emptyset$ (i.e., $1 \notin h(\tilde{A})$)
- Each element $a \in \tilde{A}$ belongs to $h^{-1}(1)$ (i.e., $h(a) = 1$) with probability $1/t$
- Thus

$$Pr = \Pr[h^{-1}(1) \cap \tilde{A} = \emptyset] = (1 - 1/t)^{|A|}$$

- If $|A| \leq t$, then $Pr \geq (1 - 1/t)^t \geq 1/4$
- If $|A| > 2t$, then $Pr < (1 - 1/t)^{2t} \leq 1/e^2$