Advanced Topics In Algorithms (ATIA)

Lecture 1

Introduction: Computing Distinct Values

(partially based on an Advanced Algorithm course by Rajeev Motwani and Nina Mishra in Stanford University)

Massive Data Sets

- Examples
  - Web (2-4 billion pages, each 1-10 KB, possibly 20TB of text)
  - Human Genome (3 billion base pairs)
  - Walmart Market-Basket Data (24 TB)
  - Sloan Digital Sky Survey (40 TB)
  - AT&T (300 million call-records per day)
- Presentation?
  - Network Access (Web)
  - Data Warehouse (Walmart)
  - Secondary Store (Human Genome)
  - Streaming (Astronomy, AT&T)

Algorithmic Problems?

- Examples
  - Statistics (median, variance, aggregates)
  - Patterns (clustering, associations, classification)
  - Query Responses (SQL, similarity)
  - Compression (storage, communication)
- Novelty?
  - Problem size – simplicity, near-linear time
  - Models – external memory, streaming
  - Scale of data – emergent behavior?

Goal: Illustrate Models

- Computational Models
  - Main memory
  - External memory
  - Streaming data
- Algorithmic Models
  - Deterministic
  - Randomized (Las Vegas versus Monte Carlo)
  - Approximation
- Measures
  - Time/Passes
  - Space

Example – Distinct Values

- Problem
  - Sequence $X = x_1, x_2, ..., x_n$
  - Domain $U = \{0, 1, 2, ..., m-1\}$
  - Determine number of distinct values in *X*, say *D(X)*
  - Note – domain could be arbitrary (e.g., text, tuples)
- Study impact of …
  - different presentation models
  - different algorithmic models
- … and thereby understand model definition

Naïve Approach

- Counter $C(i)$ for each domain value i
- Initialize counters $C(i) \leftarrow 0$
- Scan X incrementing appropriate counters
- Works in any model
- Problems?
  - Space $O(m)$, and m could be much larger than n (e.g., when counting distinct words in web crawl)
  - In fact, Time $O(m)$ – but tricks to do initialization?

Main Memory Approach

Algorithm MM

- Pick $r = \theta(n)$, hash function $h:U \rightarrow [1..r]$
- Initialize array $A[1..r]$ and $D = 0$
- For each input value x_i
  - Check if x_i occurs in list stored at $A[h(xi)]$
  - If not, $D \leftarrow D+1$ and add x_i to list at $A[h(xi)]$
- Output D

- For "random" h, few collisions & most list-sizes $O(1)$
- Thus
  - Time $O(n)$ [Expected]
  - Space $O(n)$

Randomized Algorithms

- Las Vegas (preceding algorithm)
  - always produces right answer
  - running-time is random variable->probably fast
- Monte Carlo (will see later)
  - running-time is deterministic
  - may produce wrong answer (bounded probability)->probably correct
- Atlantic City (sometimes also called M.C.)
  - worst of both worlds->probably fast, probably correct

Types of Random Algorithms

- Monte Carlo
  - Running time bounded by input size, but answer may be wrong
  - Decision problems: If there is no solution, always returns “no”. If there is a solution, finds it with some probability >= ½.
  - Value problems: run for a bounded number of steps, produce an answer that is correct approximation with a bounded probability (function of number of steps)

Types of Random Algorithms

- Las Vegas
  - Guaranteed to produce correct answer, but running time is probabilistic
- Atlantic City
  - Running time bounded by input
  - Can return either “yes” or “no” regardless of correct answer. Correct with probability >= 2/3.

External Memory Model

- Required when input X doesn't fit in memory
- M words of memory
- Input size $n >> M$
- Data stored on disk
  - Disk block size $B << M$
  - Unit time to transfer disk block to memory
- Memory operations are free

Justification?

- Block read/write?
  - Transfer rate ≈ 100 MB/sec (say)
  - Block size ≈ 100 KB (say)
  - Block transfer time << Seek time
  - Thus – only count number of seeks
- Linear Scan
  - even better as avoids random seeks
- Free memory operations?
  - Processor speeds – multi-GHz
  - Disk seek time ≈ 0.01 sec

External Memory Algorithm?

- Question – Why not just use Algorithm MM?
- Problem
  - Array A does not fit in memory
  - For each value, need a random portion of A
  - Each value involves a disk block read
  - Thus – $\Omega(n)$ disk block accesses
- Linear time – $O(n/B)$ in this model

Algorithm EM

- Merge Sort
  - Partition into *M/B* groups
  - Sort each group (recursively)
  - Merge groups using *n/B* block accesses
    (need to hold 1 block from each group in memory)
- Compute *D(X)* – one more pass
- Sorting Time – $\frac{n}{B}\log_{M/B} n$
- Total Time – $\frac{n}{B}(1+\log_{\frac{M}{B}} n)$

Problem with Algorithm EM

- Need to sort and reorder blocks on disk
- Databases
  - Tuples with multiple attributes
  - Data might need to be ordered by attribute Y
  - Algorithm EM reorders by attribute X
- In any case, sorting is too expensive
- Alternate Approach
  - Sample portions of data
  - Use sample to estimate distinct values

Sampling-based Approaches

- Benefit – sublinear space
- Cost – estimation error
- Naïve approach
  - Random Sample R (of size r) of n values in X
  - Compute D(R)
  - Estimator $\hat{D} = D(R) \times n / r$
- Error – low-frequency value underrepresented
- Existence of less naïve approaches?

Negative Result for Sampling

[Charikar, Chaudhuri, Motwani, Narasayya 2000]

Theorem: Let E be estimator for D(X) examining r<n values in X, possibly in an adaptive and randomized order. Then, for any $\delta > e^{-r}$, E must have relative error

$$\sqrt{\frac{n-r}{2r}\ln\frac{1}{\delta}}$$

with probability at least δ.

- Example
  - Say, r = n/5
  - Error 20% with probability 1/2

Scenario Analysis

Scenario A:

- all values in X are identical (say V)
- D(X) = 1

Scenario B:

- distinct values in X are {V, W1, …, Wk},
- V appears *n-k* times
- each Wi appears once
- Wi's are randomly distributed
- *D(X) = k+1*

Proof

- Little Birdie – one of Scenarios A or B only
- Suppose
  - E examines elements X(1), X(2), …, X(r) in that order
  - choice of X(i) could be randomized and depend arbitrarily on values of X(1), …, X(i-1)
- Lemma

  P[X(i)=V | X(1)=X(2)=…=X(i-1)=V] $\geq \frac{n-i-k+1}{n-i+1}$
- Why?
  - No information on whether Scenario A or B
  - Wi values are randomly distributed

Proof (continued)

- Define $\mathcal{EV}$ – event {X(1)=X(2)=…=X(r)=V}

$$\mathrm{P}[\mathrm{E}\mathcal{V}] = \prod_{i=1}^{r} \mathrm{P}[X(i) = V \mid X(1) = X(2) = \ldots = X(i-1) = V]$$

$$\geq \prod_{i=1}^{r} \frac{n-i-k+1}{n-i+1} \geq \left(\frac{n-r-k}{n-r}\right)^{r}$$

$$= \left(1 - \frac{k}{n-r}\right)^{r} \geq \exp\left(\frac{-2kr}{n-r}\right)$$

- Last inequality because

$$1 - Z \geq \exp(-2Z), \quad \text{for } 0 \leq Z \leq 1/2$$

Proof (conclusion)

- Choose $k = \frac{n-r}{2r} \ln \frac{1}{\delta}$ to obtain $\mathrm{P}[\mathcal{EV}] \geq \delta$
- Thus:
  - Scenario A → $\mathrm{P}[\mathcal{EV}] = 1$
  - Scenario B → $\mathrm{P}[\mathcal{EV}] \geq \delta$
- Suppose
  - E returns estimate Z when $\mathcal{EV}$ happens
  - Scenario A → D(X)=1
  - Scenario B → D(X)=k+1
  - Z must have worst-case error > $\sqrt{k}$

Streaming Model

- Motivating Scenarios
  - Data flowing directly from generating source
  - “Infinite” stream cannot be stored
  - Real-time requirements for analysis
  - Possibly from disk, streamed via Linear Scan
- Model
  - Stream – at each step can request next input value
  - Assume stream size n is finite/known (fix later)
  - Memory size M << n
- VERIFY – earlier algorithms not applicable

Negative Result

Theorem: Deterministic algorithms need M = $\Omega(n \log m)$

Proof:

- Assume – deterministic A uses $o(n \log m)$ bits
- Choose – input $X \subset U$ of size $n<m$
- Denote by S – state of A after X
- Can check if any $x_i \in X$ by feeding to A as next input
  - D(X) doesn't increase iff $x_i \in X$
- Information-theoretically – can recover X from S
- Thus – $\binom{m}{n}$ states require $\Omega(n \log m)$ memory bits

Randomized Approximation

(based on [Indyk-Motwani 1998])

- Lower bound does not rule out randomized or approximate solutions
- Algorithm SM – For fixed t, is D(X) >> t?
  - Choose hash function $h: U \rightarrow [1..t]$
  - Initialize answer to NO
  - For each x_i, if $h(x_i) = t$, set answer to YES
- Theorem:
  - If D(X) < t, P[SM outputs NO] > 0.25
  - If D(X) > 2t, P[SM outputs NO] < 0.136 = 1/e^2

Analysis

- Let – Y be set of distinct elements of X
- SM(X) = NO ⇔ no element of Y hashes to *t*
- P[element hashes to *t*] = *1/t*
- Thus – P[SM(X) = NO] = $(1-1/t)^{|Y|}$
- Since |Y| = D(X),
  - If D(X) < t, P[SM(X) = NO] > $(1-1/t)^{t}$ > 0.25
  - If D(X) > 2t, P[SM(X) = NO] < $(1-1/t)^{2t}$ < 1/e^2
- Observe – need 1 bit memory only!

Boosting Accuracy

- With 1 bit →
  can probabilistically distinguish D(X) < t from D(X) > 2t
- Running O(log 1/δ) instances in parallel →
  reduces error probability to any δ>0
- Running O(log n) in parallel for t = 1, 2, 4, 8 …, n →
  can estimate D(X) within factor 2
- Choice of factor 2 is arbitrary →
  can use factor (1+ε) to reduce error to ε
- EXERCISE – Verify that we can estimate D(X) within factor (1±ε) with probability (1-δ) using space

$$O\left(\log \frac{n}{\varepsilon^2} \times \log \frac{1}{\delta}\right)$$

Sampling versus Counting

- Observe
  - Count merely abstraction – need subsequent analytics
  - Data tuples – X merely one of many attributes
  - Databases – selection predicate, join results, …
  - Networking – need to combine distributed streams
- Single-pass Approaches
  - Good accuracy
  - But gives only a count -- cannot handle extensions
- Sampling-based Approaches
  - Keeps actual data – can address extensions
  - Strong negative result

Distinct Sampling for Streams

[Gibbons 2001]

- Best of both worlds
  - Good accuracy
  - Maintains "distinct sample" over stream
  - Handles distributed setting
- Basic idea
  - Hash – random "priority" for domain values
  - Tracks $O\left(\varepsilon^{-2} \log \delta^{-1}\right)$ highest priority values seen
  - Random sample of tuples for each such value
  - Relative error ε with probability $1-\delta$

Hash Function

- Domain U = [0..m-1]
- Hashing
  - Random A, B from U, with A>0
  - g(x) = Ax + B (mod m)
  - h(x) – # leading 0s in binary representation of g(x)
- Clearly – $0 \le h(x) \le \log m$
- Fact

$$\mathrm{P}\left[h(x) = l\right] = 2^{-(l+1)}$$

Overall Idea

- Hash → random "level" for each domain value
- Compute level for stream elements
- Invariant
  - Current Level – *cur_lev*
  - Sample S – all distinct values scanned so far of level at least *cur_lev*
- Observe
  - Random hash → random sample of distinct values
  - For each value → can keep sample of their tuples

Algorithm DS (Distinct Sample)

- Parameters – memory size $\mathrm{M} = O\left(\varepsilon^{-2} \log \delta^{-1}\right)$
- Initialize – *cur_lev*←0; S←empty
- For each input x
  - L ← *h(x)*
  - If L>*cur_lev* then add *x* to S
  - If |S| > M
    - delete from S all values of level *cur_lev*
    - *cur_lev* ← *cur_lev* +1
- Return $2^{cur\_lev} \times |S|$

Analysis

- Invariant – S contains all values x such that

$$h(x) \geq cur\_lev$$

- By construction

$$\mathrm{P}\left[h(x) \geq cur\_lev\right] = 2^{-cur\_lev}$$

- Thus

$$\mathrm{E}\left[\,|\,\mathrm{S}\,|\,\right] = 2^{-cur\_lev} \times D(X)$$

- EXERCISE – verify deviation bound

Summary

- Algorithmic paradigms vary dramatically over models
- Negative results abound
- Power of randomization
- Need to approximate
- References
  - *Towards Estimation Error Guarantees for Distinct Values.* Charikar, Chaudhuri, Motwani, and Narasayya. PODS 2000.
  - *Probabilistic counting algorithms for data base applications.* Flajolet and Martin. JCSS 31, 2, 1985.
  - *The space complexity of approximating the frequency moments.* Alon, Matias, and Szegedy. STOC 1996.
  - *Distinct Sampling for Highly-Accurate Answers to Distinct Value Queries and Event Reports.* Gibbons. VLDB 2001.