COT 6936: Topics in Algorithms

Giri Narasimhan

ECS 254A / EC 2443; Phone: x3748

giri@cs.fiu.edu

http://www.cs.fiu.edu/~giri/teach/COT6936_S12.html

COT 6936: Topics in Algorithms

Amortized Analysis

Amortized Analysis

- Consider (worst-case) time complexity of sequence of n operations, not cost of a single operation.
- ***Traditional Analysis:*** Cost of sequence of n operations = n S(n), where S(n) = worst case cost of each of the n operations
- ***Amortized Cost*** = T(n)/n, where T(n) = worst case total cost of n operations in the sequence.
- Amortized cost can be small even with some expensive operations. Worst case may not occur in every operation, even in worst case. Cost of operations often correlated.

Problem 1: Binary Counter

- Data Structure: binary counter b.
- Operations: Inc(b).
  - Cost of Inc(b) = number of bits flipped in the operation.
- What's the total cost of N operations when this counter counts up to integer N?
- ***Approach 1: simple analysis***
  - Size of counter is log(N). Worst case when every bit flipped. For N operations, total worst-case cost = O(Nlog(N))

Approach 2: Binary Counter

- **Intuition**: Worst case cannot happen all the time!

000000

000001

000010

000011

000100

000101

000110

000111

Bit 0 flips every time;

Bit 1 flips every other time;

Bit 2 flips every fourth time, etc…

Bit k flips every 2^k time.

So the total bits flipped in N operations, when the counter counts from 1 to N, will be = ?

$$T(N) = \sum_{k=0}^{\log N} \frac{N}{2^k} < N \sum_{k=0}^{\infty} \frac{1}{2^k} = 2N$$

So the amortized cost will be T(N)/N = 2.

Approach 3: Binary Counter

- For k bit counters, the total cost is

  $t(k) = 2 \times t(k-1) + 1$
- So for N operations, $T(N) = t(\log(N))$.
- $t(k) = ?$
- $T(N)$ can be proved to be bounded by $2N$.

Amortized Analysis: Potential Method

- For n operations, the data structure goes through states: D_0, D_1, D_2, …, D_n with costs c_1, c_2, …, c_n
- Define potential function $\Phi(D_i)$: represents the potential energy of data structure after i_{th} operation.
- The amortized cost of the i_{th} operation is defined by:

$$\boxed{\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})}$$

- The total amortized cost is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{N} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) = \Phi(D_n) - \Phi(D_0) + \sum_{i=1}^{n} c_i$$

$$\sum_{i=1}^{n} c_i = -\left(\Phi(D_n) - \Phi(D_0)\right) + \sum_{i=1}^{n} \hat{c}_i$$

Potential Method for Binary Counter

- Potential function = ??
- $\Phi(D)$ = # of 1's in counter
- Assume that in i-th iteration Inc(b) changes
  - 1 → 0 (j bits)
  - 0 → 1 (1 bit)
  - $\Phi(D_{i-1}) = k$; $\Phi(D_i) = k - j + 1$
  - Change in potential = $(k - j + 1) - k = 1-j$
  - Real cost = $j + 1$
  - Amortized cost = Real cost + change in potential
  - Amortized cost = $j + 1 - j + 1 = 2$

Problem 2: Stack Operations

- Data Structure: **Stack**
- Operations:
  - *Push(s,x)* : Push object *x* into stack *s*.
    - Cost: T(push) = O(1).
  - *Pop(s) :* Pop the top object in stack *s*.
    - Cost: T(pop) = O(1).
  - *MultiPop(s,k) ;* Pop the top *k* objects in stack *s*.
    - Cost: T(mp) = O(size(s)) worst case
- ***Assumption:*** Start with an empty stack
- ***Simple analysis:*** For N operations, maximum stack size = N. Worst-case cost of *MultiPop* = O(N). Total worst-case cost of N operations is at most N x T(mp) = O(N^2).

Amortized analysis: Stack Operations

- **Intuition**: Worst case cannot happen all the time!
- **Idea**: pay a dollar for every operation, then count carefully.
- Pay $2 for each *Push* operation, one to pay for operation, another for "future use" (pin it to object on stack).
- For *Pop* or *MultiPop*, instead of paying from pocket, pay for operations with extra dollar pinned to popped objects.
- Total cost of N operations must be less than 2 x N
- **Amortized cost** = T(N)/N = 2.

Potential Method for Stack Problem

- Potential function $\Phi(D)$ = # of items in stack
- Push
  - Change in potential = 1; Real cost = 1
  - Amortized Cost = 2
- MultiPop [Assume j items popped in i^{th} iter]
  - $\Phi(D_{i-1}) = k$; $\Phi(D_i) = k - j$
  - Real cost = j
  - Change in potential = $-j$
  - Amortized cost = Real cost + change in potential
  - Amortized cost = $j - j = 0$

Pop: $j = 1$

COT 6936: Topics in Algorithms

Streaming Algorithms

Massive Data Sets

- Examples of large persistent data sets
  - WalMart Transaction data (1 PB?)
  - Sloan Digital Sky Survey (100 TB)
  - Web (over a Trillion pages; over 1 PB of text)
  - CERN (expected to produce ~40 TB/sec)
- Large data sets with time-sensitive data
  - Financial tickers data (NASDAQ: 50K trans/s)
  - Credit Card usage traffic
  - Network Traffic: Telecom & ISP traffic
  - Sensor data

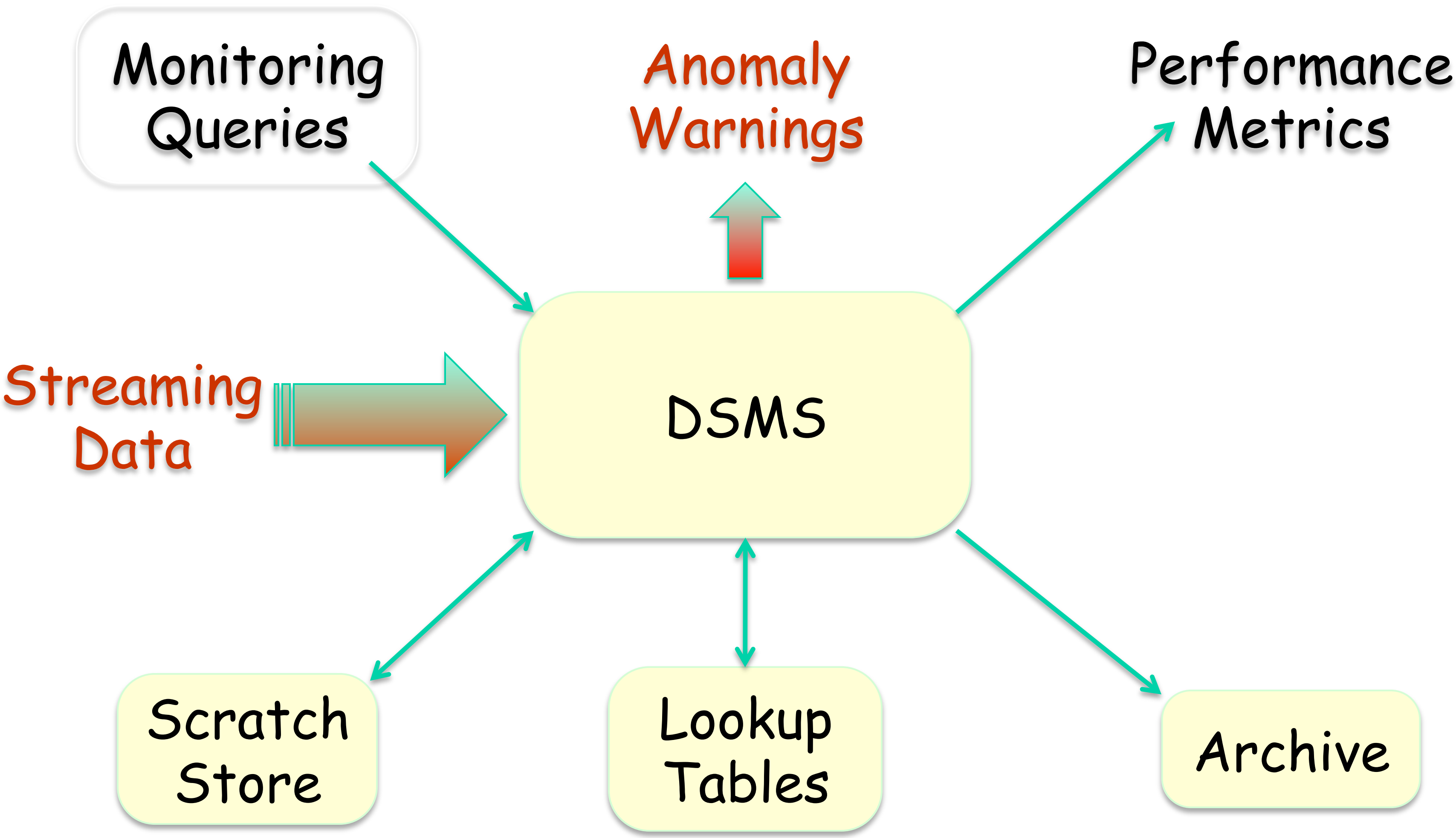
Important Issues for Stream Algorithms

- Key parameters
  - Amount of memory available; window size
  - Per-item processing time; # of Passes on data
  - Tolerance to error
- What is needed?
  - Summarizations, synposes, sketches
  - Randomization and sampling
  - Pattern Discovery
  - Anomaly Detection
  - Clustering and Classifications

Streaming Model of Computation

- N = # of items seen so far, window size
  - amount of memory available
- ε= error tolerance
- Memory usage = poly($1/\varepsilon$, $\log N$)
- Query Time = poly($1/\varepsilon$, $\log N$)

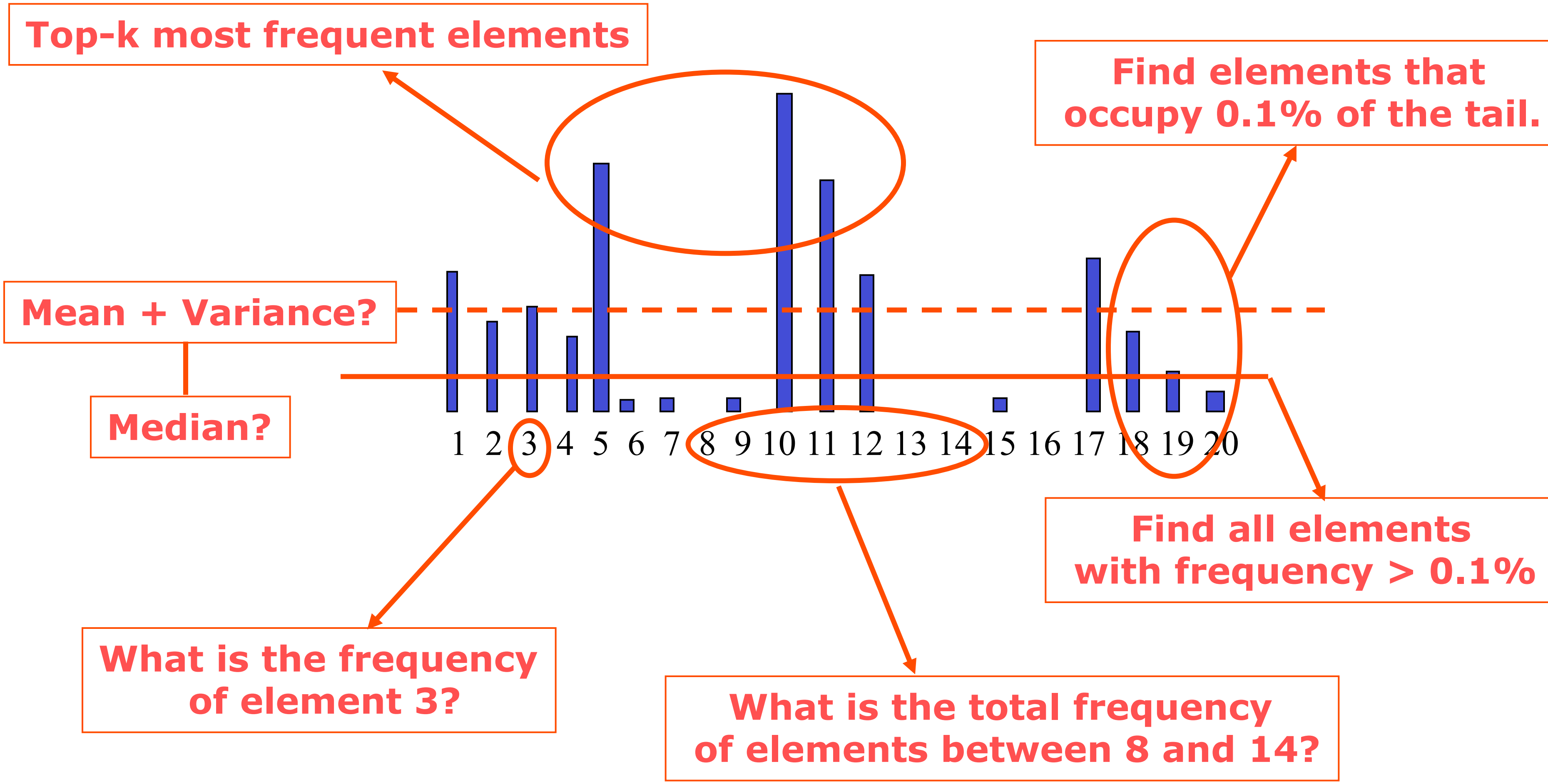
Network Monitoring System



Based on slide by R. Motwani, 2005

Frequency Related Problems

Analytics on Packet Headers – IP Addresses



How many elements have non-zero frequency?

Based on slide by R. Motwani, 2005

Warm up Problems

- Given stream of values, find mean.
  - Easy.
  - Maintain sum of all values and number of items
- Given stream, find standard deviation.
  - Not so hard
- Given stream of bits and window size N, count number of 1s in window
  - Naïve: Store the window: requires N bits
  - Can you do better?

Problem: Finding Missing Label

- Packets labeled from set $\{1,\dots,n\}$ and arrive in random order. Assume one packet is missing.
- Find the label of the missing packet.
  - Bit vector of length n
    - Space O(n)

Problem: Find Missing Label

6	4	0	1	7	2	5	28-Sum	Parity Bit
6	10	10	11	18	20	25	3	
1	1	0	0	1	0	1		0
1	0	0	0	1	1	0		1
0	0	0	1	1	0	1		1

- Maintain Sum of Labels and subtract from required sum = $\Sigma_{i=[1..n]} i = n(n+1)/2$
  - Space = $\log(n(n+1)/2) < 2 \log n$
- Optimal algorithm [Assume $n = 2^m$]
  - Store parity sum of each bit of all numbers seen
  - Missing number = Final parity sum bits

Problem: Finding Missing Numbers

- Same as problem 1, but there may be up to k missing numbers.
- Instead of sum of numbers, we maintain k different functions of the numbers seen.
  - Decoding is not so easy
    - Needs factoring polynomials
    - No known deterministic algorithms (?)
  - Randomized algorithms
    - $O(k^2 \log n)$
    - $O(k \log k \log n)$

Problem: Find number of unique items

- Simple hashing scheme to do counting
  - Space = O(m)
  - Time = O(1) per item in stream

Problem: Find fraction of rare items

- Fishing problem
  - Species $U = \{1, \ldots, u\}$
- Input stream consists of species caught and observed at time t
- Rarity: $r[t] = |\{j|\ c_t[j] = 1\}| \ / \ u$
  - Number of items in stream that are rare (i.e., appear only once in the catch so far)

Rarity Problem

- Rarity: $r[t] = |\{j|\ c_t[j] = 1\}|\ /\ u$
  - Deterministic alg: 2u bits + counter for r (Easy!)
  - If s[t] is number of species in input stream, then deterministic alg has lower bound of Ω(s[t]) bits
    - It takes Ω(s[t]) bits to keep track of which species are in the stream.
    - If deterministic alg using o(s[t]) bits, then we can keep track of species in stream with o(s[t]) bits.
      - Contradiction!
  - Randomization? Approximation?
    - Maintain info on k species; report ratio r[t,k]
    - If r[t,k] > 1/k, then r[t,k] is a good estimate for r[t]

Rarity Problem

- Randomization? Approximation?
  - Maintain info on k species; report ratio r[t,k]
  - If r[t,k] > 1/k, then r[t,k] is a good estimate for r[t]
  - Often does not work if u is very large – then modify the definition of rarity

Problem: Counting

- Given a stream of bits, at every time instant, maintain count of number of 1s in last N elements
  - Deterministic algorithms
    - $\Theta(N)$ bits of memory to answer in $O(1)$ time [Why?]

Problem: Counting

- How well can you approximate with o(N) memory? [Datar et al. SIAM J C 2002]
  - Use histogram techniques
    - Build time-based histograms in which every bucket represents a contiguous time interval
    - Idea: Use uniform buckets
    - Problem: 1s may not be distributed uniformly
    - Solution: Use non-uniform buckets
  - Results
    - $O((1/\varepsilon)\log^2 N)$ bits
    - $(1+\varepsilon)$-approximate count in $O(1)$ time

$\Omega((1/\varepsilon)\log^2(N\varepsilon))$

Other problems

- COUNTING: Given a stream of bits, at every time instant, maintain count of number of 1s in last N elements
- SUM: Given a stream of positive integers in range [0..R], at every time instant, maintain sum of last N elements

Clustering

- K-Means
  - Constant-factor approximation, $O(nk \log k)$ time, $O(k)$ space, single pass [Charikar et al. 1997]
- K-Medians
  - Constant-factor approximation, $O(nk \log k)$ time, $O(n^{\varepsilon})$ space, single pass [Guha et al. 2002]