COT 6936: Topics in Algorithms

Giri Narasimhan

ECS 254A / EC 2474; Phone x3748; Email: giri@cs.fiu.edu
HOMEPAGE: http://www.cs.fiu.edu/~giri
https://moodle.cis.fiu.edu/v2.1/course/view.php?id=612

Mar 18, 2014

Presentation Outline

Credits and Acknowledgments

Lecture slides are based on

1. A *Tutorial* by Minos Garofalakis, Johannes Gehreke, and Rajeev Rastogi, VLDB 2002. You can see the original slides at: `http://www.cse.ust.hk/vldb2002/program-info/tutorial-slides/T5garofalalis.pdf`
2. Lecture slides by Rajeev Motwani, Stanford University, See lecture15 or Handout 17 on "Streaming Data" from: `http://theory.stanford.edu/~rajeev/cs361.html`
3. Notes by M. Muthukrishnan from: `http://www.cs.mcgill.ca/~denis/notes09.pdf`

COT 6936: Topics in Algorithms

Giri Narasimhan

Presentation Outline

Applications

- Network traffic monitoring

Applications

- Network traffic monitoring
- Telecommunication call detail records

Applications

- Network traffic monitoring
- Telecommunication call detail records
- Retail transaction; ATM transactions

Applications

- Network traffic monitoring
- Telecommunication call detail records
- Retail transaction; ATM transactions
- Log records for web servers

Applications

- Network traffic monitoring
- Telecommunication call detail records
- Retail transaction; ATM transactions
- Log records for web servers
- Sensor network data

Applications

- Network traffic monitoring
- Telecommunication call detail records
- Retail transaction; ATM transactions
- Log records for web servers
- Sensor network data
- Financial market transactions data

Constraints and Goals

- Data appearing at a rapid rate

Constraints and Goals

- Data appearing at a rapid rate
- Massive volume

Constraints and Goals

- Data appearing at a rapid rate
- Massive volume
- Process queries, mine patterns, compute statistics

Constraints and Goals

- Data appearing at a rapid rate
- Massive volume
- Process queries, mine patterns, compute statistics
- Real time computations needed

Constraints and Goals

- Data appearing at a rapid rate
- Massive volume
- Process queries, mine patterns, compute statistics
- Real time computations needed
- Single pass: Allowed to see data only once

Constraints and Goals

- Data appearing at a rapid rate
- Massive volume
- Process queries, mine patterns, compute statistics
- Real time computations needed
- Single pass: Allowed to see data only once
- Limited memory to store processed data

Constraints and Goals

- Data appearing at a rapid rate
- Massive volume
- Process queries, mine patterns, compute statistics
- Real time computations needed
- Single pass: Allowed to see data only once
- Limited memory to store processed data
- Approximate answers and/or randomization may be acceptable

Constraints and Goals

- Data appearing at a rapid rate
- Massive volume
- Process queries, mine patterns, compute statistics
- Real time computations needed
- Single pass: Allowed to see data only once
- Limited memory to store processed data
- Approximate answers and/or randomization may be acceptable
- Quick responses, i.e., short query time

Big Data Sets

- Examples of large persistent data sets

Big Data Sets

- Examples of large persistent data sets
  - Walmart Transaction data (PBs)
  - Sloan Digital Sky Survey (100 TBs)
  - WWW (¿ Trillion pages)
  - CERN (40TB/sec)
- Large Data Sets with time-sensitive data

Big Data Sets

- Examples of large persistent data sets
  - Walmart Transaction data (PBs)
  - Sloan Digital Sky Survey (100 TBs)
  - WWW (¿ Trillion pages)
  - CERN (40TB/sec)
- Large Data Sets with time-sensitive data
  - Financial data (e.g. NASDAQ: 50K transactions/sec)
  - Credit Card usage traffic
  - Network Traffic: Telecommunications and ISP traffic
  - Sensor data

Presentation Outline

Warm-up Problems

- Average

Warm-up Problems

- Average
  - Easy

Warm-up Problems

- Average
  - Easy
  - Maintain sum and count of items

Warm-up Problems

- Average
  - Easy
  - Maintain sum and count of items
- Standard Deviation

Warm-up Problems

- Average
  - Easy
  - Maintain sum and count of items
- Standard Deviation
  - Not too hard ...

Warm-up Problems

- Average
  - Easy
  - Maintain sum and count of items
- Standard Deviation
  - Not too hard ...
- Count number of 1's in window of size N in a bit stream

Warm-up Problems

- Average
  - Easy
  - Maintain sum and count of items
- Standard Deviation
  - Not too hard ...
- Count number of 1's in window of size N in a bit stream
  - Store window itself: requires N bits
  - Can you do better?

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order.

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume one packet is missing.

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume one packet is missing. Find label of missing packet.

- Use bit vector of length n.

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume one packet is missing. Find label of missing packet.

- Use bit vector of length n. Space used $= O(n)$.

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume one packet is missing. Find label of missing packet.

- Use bit vector of length n. Space used $= O(n)$.
- Improved Algorithm:

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume one packet is missing. Find label of missing packet.

- Use bit vector of length n. Space used $= O(n)$.
- Improved Algorithm: Maintain sum of labels

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume one packet is missing. Find label of missing packet.

- Use bit vector of length n. Space used $= O(n)$.
- Improved Algorithm: Maintain sum of labels and subtract from required sum of $n(n+1)/2$.

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume one packet is missing. Find label of missing packet.

- Use bit vector of length n. Space used $= O(n)$.
- Improved Algorithm: Maintain sum of labels and subtract from required sum of $n(n+1)/2$. Space used $= 2 \log n$

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume one packet is missing. Find label of missing packet.

- Use bit vector of length n. Space used $= O(n)$.
- Improved Algorithm: Maintain sum of labels and subtract from required sum of $n(n+1)/2$. Space used $= 2 \log n$
- Optimal Algorithm:

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume one packet is missing. Find label of missing packet.

- Use bit vector of length n. Space used $= O(n)$.
- Improved Algorithm: Maintain sum of labels and subtract from required sum of $n(n+1)/2$. Space used $= 2 \log n$
- Optimal Algorithm:
  - Store parity sum of each bit of all numbers seen

Find Missing Label

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume one packet is missing. Find label of missing packet.

- Use bit vector of length n. Space used $= O(n)$.
- Improved Algorithm: Maintain sum of labels and subtract from required sum of $n(n+1)/2$. Space used $= 2 \log n$
- Optimal Algorithm:
  - Store parity sum of each bit of all numbers seen
  - Missing number $=$ Final parity sum

Find Missing Labels

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume up to k packets missing.

Find Missing Labels

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume up to k packets missing. Find labels of missing packets.

- Maintain k different functions of numbers seen.

Find Missing Labels

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume up to k packets missing. Find labels of missing packets.

- Maintain k different functions of numbers seen.
- Decoding:

Find Missing Labels

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume up to k packets missing. Find labels of missing packets.

- Maintain k different functions of numbers seen.
- Decoding: Not easy

Find Missing Labels

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume up to k packets missing. Find labels of missing packets.

- Maintain k different functions of numbers seen.
- Decoding: Not easy and needs factoring polynomials

Find Missing Labels

Packets labeled from set $\{1, \ldots, n\}$ and arrive in random order. Assume up to k packets missing. Find labels of missing packets.

- Maintain k different functions of numbers seen.
- Decoding: Not easy and needs factoring polynomials

Presentation Outline

Network Traffic Monitoring

- Monitor link bandwidth usage, estimate traffic demands
- Quickly detect faults, congestion, and other causes
- Load balancing, improved resource allocation
- Detect anomalies in traffic, spikes, etc.

Network Traffic Monitoring

- IP session data (collected using Cisco NetFlow)

Source	Destination	Duration	Bytes	Protocol
10.1.0.2	16.2.3.7	12	20K	http
18.6.7.1	12.4.0.3	16	24K	http
13.9.4.3	11.6.8.2	15	20K	http
15.2.2.9	17.1.2.1	19	40K	http
12.4.3.8	14.8.7.4	26	58K	http
10.5.1.3	13.0.0.1	27	100K	ftp
11.1.0.6	10.3.4.5	32	300K	ftp
19.7.1.2	16.5.5.8	18	80K	ftp

- AT&T collects 100 GBs of NetFlow data each day

Traffic Questions

See `http://www.cs.fiu.edu/~giri/teach/6936/S14/LecX1_StreamQuestions.pdf`

Network Traffic Monitoring

- Traffic Volume Estimates

Network Traffic Monitoring

- Traffic Volume Estimates
  - Volume between specific pairs of IP addresses?
  - Active IP addresses; top 100 active IP addresses
  - Avg durection and # of bytes per session
- Anomaly/Fraud Detection and Security issues

Network Traffic Monitoring

- Traffic Volume Estimates
  - Volume between specific pairs of IP addresses?
  - Active IP addresses; top 100 active IP addresses
  - Avg durection and # of bytes per session
- Anomaly/Fraud Detection and Security issues
  - Large volume or duration sessions
  - Sessions with spikes of traffic
  - IP addresses involved in long sessions
- Deterministic vs Randomized Approaches

Network Traffic Monitoring

- Traffic Volume Estimates
  - Volume between specific pairs of IP addresses?
  - Active IP addresses; top 100 active IP addresses
  - Avg durection and # of bytes per session
- Anomaly/Fraud Detection and Security issues
  - Large volume or duration sessions
  - Sessions with spikes of traffic
  - IP addresses involved in long sessions
- Deterministic vs Randomized Approaches
  - With limited memory, deterministic methods can only compute approximate answers
  - Randomized methods compute approx answers w.h.p.

Presentation Outline

- **Sampling**

- **Sampling**
  - Pick a random sample and apply query to it

- **Sampling**
  - Pick a random sample and apply query to it
- Example: `select func from` R `where` $R.e$ `is odd`

- **Sampling**
  - Pick a random sample and apply query to it
- Example: `select func from` R `where` $R.e$ `is odd`
  - Data Stream, R:

9	3	5	2	7	1	6	5	8	4	9	1

- **Sampling**
  - Pick a random sample and apply query to it
- Example: `select` `func` `from` R `where` $R.e$ `is odd`
  - Data Stream, R:

9	3	5	2	7	1	6	5	8	4	9	1

  - Randomly sample:

9	3	5	2	7	1	6	5	8	4	9	1

- **Sampling**
  - Pick a random sample and apply query to it
- Example: `select func from` R `where` $R.e$ `is odd`
  - Data Stream, R:

9	3	5	2	7	1	6	5	8	4	9	1

  - Randomly sample:

9	3	5	2	7	1	6	5	8	4	9	1

  - Sample, S:

9	5	1	8

Randomized Algorithms for Streaming Data

- **Sampling**
  - Pick a random sample and apply query to it
- Example: `select func from` R `where` $R.e$ `is odd`
  - Data Stream, R:

9	3	5	2	7	1	6	5	8	4	9	1

  - Randomly sample:

9	3	5	2	7	1	6	5	8	4	9	1

  - Sample, S:

9	5	1	8

- If func is avg,

- **Sampling**
  - Pick a random sample and apply query to it
- Example: `select` `func` `from` R `where` $R.e$ `is odd`
  - Data Stream, R:

9	3	5	2	7	1	6	5	8	4	9	1

  - Randomly sample:

9	3	5	2	7	1	6	5	8	4	9	1

  - Sample, S:

9	5	1	8

- If func is avg, then return average of odd items in S, i.e., 5

Randomized Algorithms for Streaming Data

- **Sampling**
  - Pick a random sample and apply query to it
- Example: `select func from` R `where` $R.e$ `is odd`
  - Data Stream, R:

9	3	5	2	7	1	6	5	8	4	9	1

  - Randomly sample:

9	3	5	2	7	1	6	5	8	4	9	1

  - Sample, S:

9	5	1	8

- If func is avg, then return average of odd items in S, i.e., 5
- If func is count,

Randomized Algorithms for Streaming Data

- **Sampling**
  - Pick a random sample and apply query to it
- Example: `select func from` R `where` $R.e$ `is odd`
  - Data Stream, R:

9	3	5	2	7	1	6	5	8	4	9	1

  - Randomly sample:

9	3	5	2	7	1	6	5	8	4	9	1

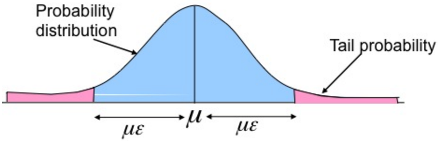
  - Sample, S:

9	5	1	8

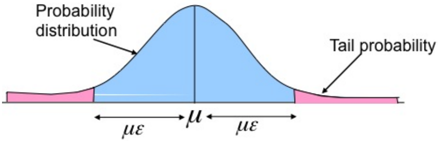
- If func is avg, then return average of odd items in S, i.e., 5
- If func is count, then return count of odd items in S, scaled for length of sequence, i.e., $3 * (12/4) = 9$

How to guarantee error estimates of answers?

- **Tools for Tail Inequalities**:



- **Tools for Tail Inequalities**:

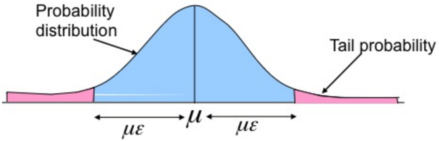


Let X be a r.v., $\mu = E[X]$

- Markov inequality $Pr(X \geq \epsilon) \leq \frac{\mu}{\epsilon}$

How to guarantee error estimates of answers?

- **Tools for Tail Inequalities**:

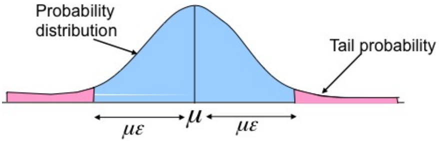


Let X be a r.v., $\mu = E[X]$

- Markov inequality $Pr(X \geq \epsilon) \leq \frac{\mu}{\epsilon}$
- Chebyshev Inequality $Pr(|X - \mu| \geq \mu\epsilon) \leq \frac{Var[X]}{\mu^2\epsilon^2}$

How to guarantee error estimates of answers?

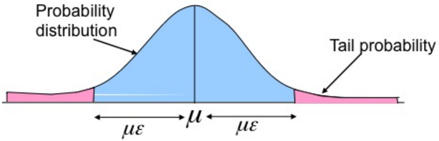
- **Tools for Tail Inequalities**:



Let X be a r.v., $\mu = E[X]$

- Markov inequality $Pr(X \geq \epsilon) \leq \frac{\mu}{\epsilon}$
- Chebyshev Inequality $Pr(|X - \mu| \geq \mu\epsilon) \leq \frac{Var[X]}{\mu^2\epsilon^2}$
- Hoeffding inequality: Good for avg. Given r.v. $X_i \in [0..r], i = 1, \ldots, m$ with mean $\overline{X}$, and any $\epsilon > 0$, $Pr(|\overline{X} - \mu|) \geq \epsilon) \geq 2e^{-2me^2/r^2}$.

- **Tools for Tail Inequalities**:



Let X be a r.v., $\mu = E[X]$

- Markov inequality $Pr(X \geq \epsilon) \leq \frac{\mu}{\epsilon}$
- Chebyshev Inequality $Pr(|X - \mu| \geq \mu\epsilon) \leq \frac{Var[X]}{\mu^2\epsilon^2}$
- Hoeffding inequality: Good for avg. Given r.v. $X_i \in [0..r], i = 1, \ldots, m$ with mean $\overline{X}$, and any $\epsilon > 0$, $Pr(|\overline{X} - \mu|) \geq \epsilon) \geq 2e^{-2me^2/r^2}$.
- Chernoff bound Good for counts. Given m independent Bernoulli trials with $Pr(X_i = 1) = p$, and $X = \sum X_i$, then $\mu = mp = E[X]$ and $Pr(|X - \mu| \geq \mu\epsilon) \leq 2e^{-\mu\epsilon^2/2}$.

How to Sample

- Reservoir Sampling [Waterman; See Vitter, ACM TOMS, 1985]

How to Sample

- Reservoir Sampling [Waterman; See Vitter, ACM TOMS, 1985]
  - How to efficiently sample n items from a stream of N items with $O(1)$ space and in single pass when N is unknown?

How to Sample

- Reservoir Sampling [Waterman; See Vitter, ACM TOMS, 1985]
  - How to efficiently sample n items from a stream of N items with $O(1)$ space and in single pass when N is unknown?
  - Reservoir algorithms select sample of size $\geq n$ and then generate sample of size n from it.

How to Sample

- Reservoir Sampling [Waterman; See Vitter, ACM TOMS, 1985]
  - How to efficiently sample n items from a stream of N items with $O(1)$ space and in single pass when N is unknown?
  - Reservoir algorithms select sample of size $\geq n$ and then generate sample of size n from it.
  - Add each new element to S with prob n/N, where $N =$ number of stream elements seen.

How to Sample

- Reservoir Sampling [Waterman; See Vitter, ACM TOMS, 1985]
  - How to efficiently sample n items from a stream of N items with $O(1)$ space and in single pass when N is unknown?
  - Reservoir algorithms select sample of size $\geq n$ and then generate sample of size n from it.
  - Add each new element to S with prob n/N, where $N =$ number of stream elements seen.
  - To *evict*, skip random number of items and replace item at that location.

How to Sample

- Reservoir Sampling [Waterman; See Vitter, ACM TOMS, 1985]
  - How to efficiently sample n items from a stream of N items with $O(1)$ space and in single pass when N is unknown?
  - Reservoir algorithms select sample of size $\geq n$ and then generate sample of size n from it.
  - Add each new element to S with prob n/N, where $N =$ number of stream elements seen.
  - To *evict*, skip random number of items and replace item at that location.

Presentation Outline

Synopses using Probabilistic Counting

To compute *k* most frequent values:

Synopses using Probabilistic Counting

To compute k most frequent values: Also called Top-k, Hotlist, Most popular list, etc.

- Adversary model can always force wrong answers

Synopses using Probabilistic Counting

To compute ***k*** **most frequent values**: Also called Top-k, Hotlist, Most popular list, etc.

- Adversary model can always force wrong answers
- Footprint refers to amount of memory used;

Synopses using Probabilistic Counting

To compute k most frequent values: Also called Top-k, Hotlist, Most popular list, etc.

- Adversary model can always force wrong answers
- Footprint refers to amount of memory used; Larger footprint, greater accuracy;

Synopses using Probabilistic Counting

To compute *k* most frequent values: Also called Top-k, Hotlist, Most popular list, etc.

- Adversary model can always force wrong answers
- Footprint refers to amount of memory used; Larger footprint, greater accuracy; Footprint assumed to be bounded

Synopses using Probabilistic Counting

To compute k most frequent values: Also called Top-k, Hotlist, Most popular list, etc.

- Adversary model can always force wrong answers
- Footprint refers to amount of memory used; Larger footprint, greater accuracy; Footprint assumed to be bounded
- Let T be estimated frequency of least frequent item in Hotlist

Synopses using Probabilistic Counting

To compute ***k*** **most frequent values**: Also called Top-k, Hotlist, Most popular list, etc.

- Adversary model can always force wrong answers
- Footprint refers to amount of memory used; Larger footprint, greater accuracy; Footprint assumed to be bounded
- Let T be estimated frequency of least frequent item in Hotlist
- Add new item to S with probability $1/T$.

Synopses using Probabilistic Counting

To compute *k* most frequent values: Also called Top-k, Hotlist, Most popular list, etc.

- Adversary model can always force wrong answers
- Footprint refers to amount of memory used; Larger footprint, greater accuracy; Footprint assumed to be bounded
- Let T be estimated frequency of least frequent item in Hotlist
- Add new item to S with probability $1/T$.
- Of T occurrences of an item, at least one will get on sample
-

For $x \in S$, $EstimatedFreq(x) = Count(x) + 0.418 \cdot T$

Synopses using Concise Sampling

To compute k most frequent values:

Synopses using Concise Sampling

To compute **k most frequent values**:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs

Synopses using Concise Sampling

To compute k **most frequent values**:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs
- For item s_i, if $s_i \in S$, increment its count;

Synopses using Concise Sampling

To compute k most frequent values:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs
- For item s_i, if $s_i \in S$, increment its count; Otherwise, add to S with probability $1/T$.

Synopses using Concise Sampling

To compute k **most frequent values**:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs
- For item s_i, if $s_i \in S$, increment its count; Otherwise, add to S with probability $1/T$.
- If size of sample exceeds M, select new threshold $T' > T$;

Synopses using Concise Sampling

To compute k most frequent values:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs
- For item s_i, if $s_i \in S$, increment its count; Otherwise, add to S with probability $1/T$.
- If size of sample exceeds M, select new threshold $T' > T$;
  - Goal: Evict each of M items with prob T/T', with preference to lower count items

Synopses using Concise Sampling

To compute k most frequent values:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs
- For item s_i, if $s_i \in S$, increment its count; Otherwise, add to S with probability $1/T$.
- If size of sample exceeds M, select new threshold $T' > T$;
  - Goal: Evict each of M items with prob T/T', with preference to lower count items
  - For each value (with count C) in S, decrement count in repeated tries until C tries or a try in which count is not decremented;

Synopses using Concise Sampling

To compute k most frequent values:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs
- For item s_i, if $s_i \in S$, increment its count; Otherwise, add to S with probability $1/T$.
- If size of sample exceeds M, select new threshold $T' > T$;
  - Goal: Evict each of M items with prob T/T', with preference to lower count items
  - For each value (with count C) in S, decrement count in repeated tries until C tries or a try in which count is not decremented;
  - First try, decrement count with probability $1 - T/T$;

Synopses using Concise Sampling

To compute k **most frequent values**:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs
- For item s_i, if $s_i \in S$, increment its count; Otherwise, add to S with probability $1/T$.
- If size of sample exceeds M, select new threshold $T' > T$;
  - Goal: Evict each of M items with prob T/T', with preference to lower count items
  - For each value (with count C) in S, decrement count in repeated tries until C tries or a try in which count is not decremented;
  - First try, decrement count with probability $1 - T/T$; Subsequent tries, decrement count with probability $1 - 1/T$;

Synopses using Concise Sampling

To compute k **most frequent values**:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs
- For item s_i, if $s_i \in S$, increment its count; Otherwise, add to S with probability $1/T$.
- If size of sample exceeds M, select new threshold $T' > T$;
  - Goal: Evict each of M items with prob T/T', with preference to lower count items
  - For each value (with count C) in S, decrement count in repeated tries until C tries or a try in which count is not decremented;
  - First try, decrement count with probability $1 - T/T$; Subsequent tries, decrement count with probability $1 - 1/T$;

Synopses using Concise Sampling

To compute ***k*** **most frequent values**:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs
- For item s_i, if $s_i \in S$, increment its count; Otherwise, add to S with probability $1/T$.
- If size of sample exceeds M, select new threshold $T' > T$;
  - Goal: Evict each of M items with prob T/T', with preference to lower count items
  - For each value (with count C) in S, decrement count in repeated tries until C tries or a try in which count is not decremented;
  - First try, decrement count with probability $1 - T/T$; Subsequent tries, decrement count with probability $1 - 1/T$;
- Subject subsequent items to higher threshold T'

Synopses using Concise Sampling

To compute k most frequent values:

- Store sample S as a set of $\langle$value, count$\rangle$ pairs
- For item s_i, if $s_i \in S$, increment its count; Otherwise, add to S with probability $1/T$.
- If size of sample exceeds M, select new threshold $T' > T$;
  - Goal: Evict each of M items with prob T/T', with preference to lower count items
  - For each value (with count C) in S, decrement count in repeated tries until C tries or a try in which count is not decremented;
  - First try, decrement count with probability $1 - T/T$; Subsequent tries, decrement count with probability $1 - 1/T$;
- Subject subsequent items to higher threshold T'

For $x \in S$, $EstimatedFreq(x) = Count(x) + 0.418 \cdot T$

Histograms

- How do we compute the frequency distribution of element values in a stream?

Histograms

- How do we compute the frequency distribution of element values in a stream?
- Histograms are basically approximate frequency distributions

Histograms

- How do we compute the frequency distribution of element values in a stream?
- Histograms are basically approximate frequency distributions
- Histograms involve partitioning the range of values into buckets and keeping track of counts in each bucket

Histograms

- How do we compute the frequency distribution of element values in a stream?
- Histograms are basically approximate frequency distributions
- Histograms involve partitioning the range of values into buckets and keeping track of counts in each bucket
- How do we compute histograms?

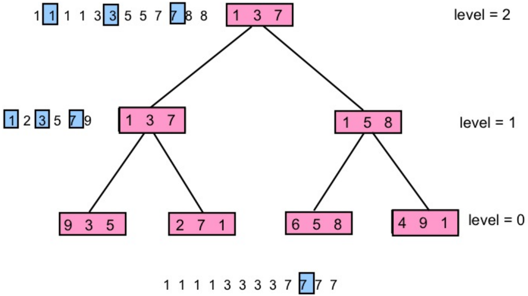
Histograms

- How do we compute the frequency distribution of element values in a stream?
- Histograms are basically approximate frequency distributions
- Histograms involve partitioning the range of values into buckets and keeping track of counts in each bucket
- How do we compute histograms? approximate quantiles?

Histograms

- How do we compute the frequency distribution of element values in a stream?
- Histograms are basically approximate frequency distributions
- Histograms involve partitioning the range of values into buckets and keeping track of counts in each bucket
- How do we compute histograms? approximate quantiles?
    - Algorithms exist to compute items with rank $(\phi \pm \epsilon)n$

Computing Quantiles in Single Pass



Miscellaneous Problems

- Clustering from streaming data

Miscellaneous Problems

- Clustering from streaming data
- Decision Trees

Miscellaneous Problems

- Clustering from streaming data
- Decision Trees
- Second Moments

Miscellaneous Problems

- Clustering from streaming data
- Decision Trees
- Second Moments
- Multi-dimensional Histograms

Miscellaneous Problems

- Clustering from streaming data
- Decision Trees
- Second Moments
- Multi-dimensional Histograms
- Number of Distinct Values; Rarity

Miscellaneous Problems

- Clustering from streaming data
- Decision Trees
- Second Moments
- Multi-dimensional Histograms
- Number of Distinct Values; Rarity
- Joins

Miscellaneous Problems

- Clustering from streaming data
- Decision Trees
- Second Moments
- Multi-dimensional Histograms
- Number of Distinct Values; Rarity
- Joins
- Self-similarity, anomalies, long-range dependence

Miscellaneous Problems

- Clustering from streaming data
- Decision Trees
- Second Moments
- Multi-dimensional Histograms
- Number of Distinct Values; Rarity
- Joins
- Self-similarity, anomalies, long-range dependence
- . . .

Presentation Outline

Stream Processing Systems

- Systems: Aurora (Brandies, Brown, MIT); NIagara (Wisconsin); STREAM (Stanford); Telegraph (Berkeley); Gigascope, Hancock, Tangram, Tapestry, Telegraph, Tribeca, ...
- System Architectures, Query Languages, Algorithms, ...