IIT KHARAGPUR

IIT GANDHINAGAR

NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Scaling Data Science

Lecture 6: Introduction to Hashing

Anirban Dasgupta

Computer Science and Engineering

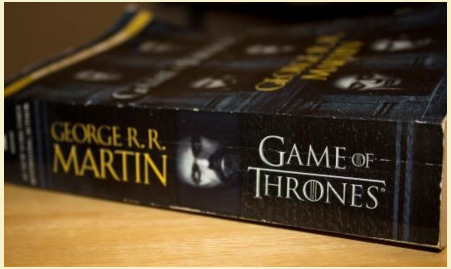
IIT GANDHINAGAR

Outline

- Outline:
  - Hash tables and hash functions
  - Universal hashing
  - Chaining
  - Multiplicative hashing

Querying

Present?



Naïve algorithm: linear in dataset size

Hash Table

- Elements come from universe U, but we need to store only n items, $n < |U|$
- Hash table
  - array of size m
  - Hash function $h: U \rightarrow \{0,1, \dots m-1\}$
- We typically use $m \ll |U|$ as well as $m < n$
  - Collisions happen when $x \neq y$, but $h(x) = h(y)$

Hash functions

- In theory, we design for worst-case behaviour of data
  - Need to choose hash function “randomly”
- Hash family $H = \{h_1, h_2, \ldots\}$
  - When creating hash table, a **single** function $h \in H$ is chosen randomly
  - We then analyse the expected query time
- However…

Hash functions

- In theory, we design for worst-case behaviour of data
  – Need to choose hash function “randomly”
- Hash family $H = \{h_1, h_2, \ldots\}$
  – When creating hash table, a **single** function $h \in H$ is chosen randomly
  – We then analyse the expected query time
- Since the algo has to carry around the “description” of the hash function, it needs $\log(|H|)$ bits of storage
  – $|H|$ cannot too big, in particular, it cannot be the set $[m]^U$, all possible functions

Hash family

- We need to create small hash families H such that choosing from it gives a function with "good behaviour"

Hash family

- We need to create small hash families H such that choosing from it gives a function with "good behaviour"
- Uniform: $\Pr_{h \in H}[h(x) = i] = \frac{1}{m}$ for all x and i

Hash family

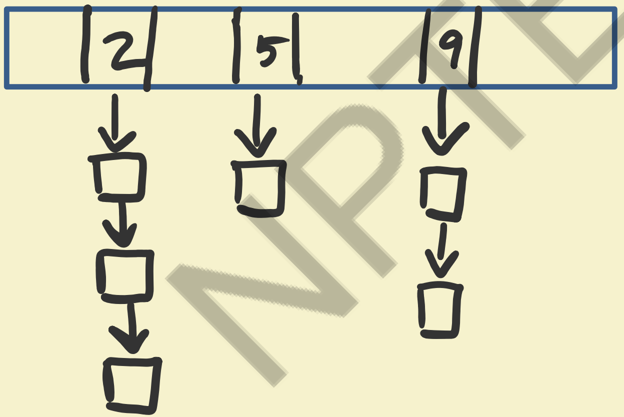
- We need to create small hash families H such that choosing from it gives a function with "good behaviour"
- Uniform: $\Pr_{h \in H}[h(x) = i] = \frac{1}{m}$ for all x and i
  - Not enough
- Universal: $\Pr_{h}[h(x) = h(y)] = \frac{1}{m}$ for all $x \neq y$

Hash family

- We need to create small hash families H such that choosing from it gives a function with "good behaviour"
- Uniform: $\Pr_{h \in H}[h(x) = i] = \frac{1}{m}$ for all x and i
  - Not enough
- Universal: $\Pr_{h}[h(x) = h(y)] = \frac{1}{m}$ for all $x \neq y$
- Near Universal: $\Pr_{h}[h(x) = h(y)] \leq \frac{2}{m}$ for all $x \neq y$

Chaining

- When collisions happen, we store elements using a linked list from that location



Chaining

- When collisions happen, we store elements using a linked list from that location
- $l(x) =$ length of chain at position $h(x)$
- Expected time to query $x = O(1 + E_h[l(x)])$
  - Same for insert and delete

Analyzing chaining

- Need to bound $E_h[l(x)]$
- For $x \neq y$, define $C_{xy} = \begin{cases} 1 \ \ if \ h(x) = h(y) \\ 0 \qquad\qquad\quad else \end{cases}$

Analyzing chaining

- Need to bound $E_h[l(x)]$
- For $x \neq y$, define $C_{xy} = \begin{cases} 1 \ \ if\ h(x) = h(y) \\ 0 \qquad\qquad else \end{cases}$
- $E_h[l(x)] = E_h[\sum_y C_{xy}]$

Analyzing chaining: universal hashing

- Need to bound $E_h[l(x)]$

- For $x \neq y$, define $C_{xy} = \begin{cases} 1 \ if \ h(x) = h(y) \\ 0 \quad\quad\quad\quad else \end{cases}$

- $E_h[l(x)] = E_h[\sum_y C_{xy}] = \sum_y \Pr[h(x) = h(y)] = \frac{n}{m}$

universal

Multiplicative hashing

- How to design small + universal hash family?
- Prime multiplicative hashing:
  - Fix a prime number $p > |U|$
  - $H = \{ h_a(x) = (ax \bmod p) \bmod m \, , a \in \{1, \ldots p-1\}\}$
  - Choosing a hash function is same as choosing $a \in \{1, \ldots p-1\}$

Multiplicative hashing

- $H = \{ h_a(x) = (ax \, mod \, p) \, mod \, m \, , a \in \{1, \ldots p-1\}\}$

- This family satisfies $\Pr_h[h(x) = h(y)] \leq \frac{1}{m}$

- Intuition: $h_a(x) \; - h_a(y) = (a(x \; - y) \, mod \, p) mod \, m$
- There are at most $\frac{p-1}{m}$ values in $\{1, .. p-1\}$ that are divisible by m

Multiplicative hashing

- $H = \{ h_a(x) = (ax \bmod p) \bmod m \, , a \in \{1, \ldots p-1\}\}$

- This family satisfies $\Pr_h[h(x) = h(y)] \leq \frac{1}{m}$

- Intuition: $h_a(x) - h_a(y) = (a(x - y) \bmod p) \bmod m$
- There are at most $\frac{p-1}{m}$ values in $\{1, .. p-1\}$ that are divisible by m
- What is the probability of choosing a such that $(a(x - y) \bmod p)$ is one of these numbers?

A property of prime numbers

WLOG $x - y \in [1, p-1]$

Property: For every $t, z \in [1, p-1]$ there exists unique $a \in [1, p-1]$ such that $az \bmod p = t$

This would imply that probability of choosing collision-causing a

$$\leq \frac{p-1}{m} \times \frac{1}{p-1} = \frac{1}{m}$$

A property of prime numbers

WLOG $x - y \in [1, p-1]$

Property: For every $t, z \in [1, p-1]$ there exists unique $a \in [1, p-1]$ such that $az \bmod p = t$

By contradiction. If not, then $\exists a, b \in [1, p-1]$ such that $(a-b)z \bmod p = 0$.

But this cannot be as p is prime.

k-wise universal

- For any distinct $(x_1, \ldots, x_k)$ and any (not necessarily distinct) $(y_1, \ldots y_k)$,

$$\Pr[h(x_1) = y_1 \wedge \cdots h(x_k) = y_k] = m^{-k}$$

- Needs only $O(k \log n)$ bit of storage

Summary

- Hashing
  - Simple and versatile
  - Main issue is design of good hash functions, much researched area
  - (near) universality guarantees small chain sizes
  - Other alternatives to chaining exist, e.g. open addressing, cuckoo hashing

References:

- Primary reference for this lecture
  - Algorithms and models of computation by Jeff Erickson: http://jeffe.cs.illinois.edu/teaching/algorithms/
- Others
  - Algorithms, by Cormen, Leiserson and Rivest
  - Randomized Algorithms by Mitzenmacher and Upfal.

Thank You!!

IIT KHARAGPUR
IIT GANDHINAGAR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Scalable Data Science

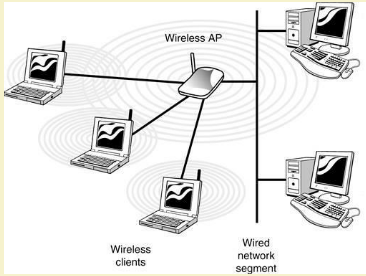
Lecture 7: Bloom Filters

Anirban Dasgupta
Computer Science and Engineering
IIT GANDHINAGAR

IIT Gandhinagar
Indian Institute of Technology Gandhinagar

Querying

ISBN present in collection?



IP seen by switch?

10.0.21.102

Solutions

- Universe U, but need to store a set of n items, $n \ll |U|$
- Hash table of size m:
  - Space $O(n \log |U|)$
  - Query time $O\left(\frac{n}{m}\right)$

Solutions

- Universe U, but need to store a set of n items, $n \ll |U|$
- Hash table of size m:
  - Space $O(n \log |U|)$
  - Query time $O\left(\frac{n}{m}\right)$
- Bit array of size $|U|$
  - Space = $|U|$
  - Query time $O(1)$

Querying, Monte Carlo style

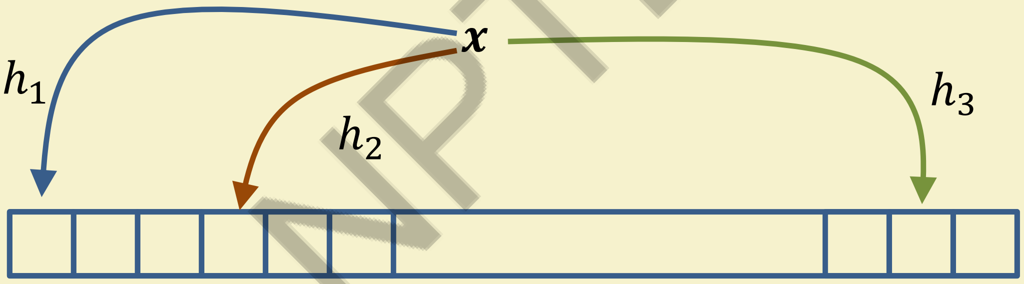
- In hash table construction, we used random hash functions
  - we never return incorrect answer
  - query time is a random variable
  - These are Las Vegas algorithms

- In Monte-Carlo randomized algorithms, we are allowed to return incorrect answers with (small) probability, say, δ

Bloom filter

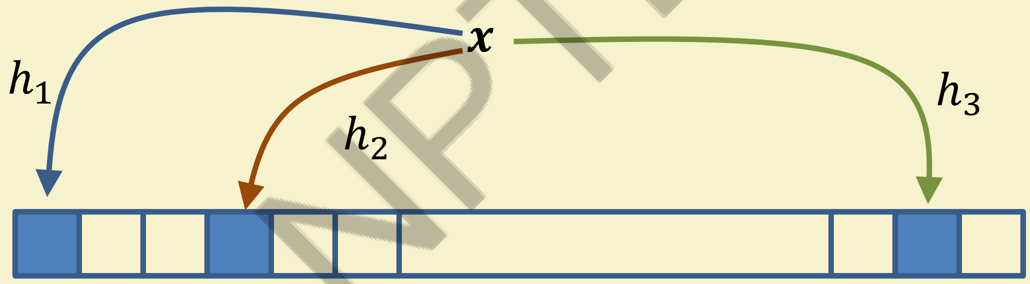
[Bloom, 1970]

- A bit-array B, $|B| = m$
- k hash functions, $h_1, h_2, \dots, h_k$, each $h_i \in U \to [m]$



Bloom filter

- A bit-array B, $|B| = m$
- k hash functions, $h_1, h_2, \ldots, h_k$, each $h_i \in U \to [m]$



Operations

- $Initialize(B)$
  - for $i \in \{1,..m\}$, $B[i] = 0$

- $Insert\ (B, x)$
  - for $i \in \{1,..k\}$, $B[h_i(x)] = 1$

- $Lookup\ (B, x)$
  - If $\wedge_{i \in \{1,\dots k\}} B[h_i(x)]$, return PRESENT, else ABSENT

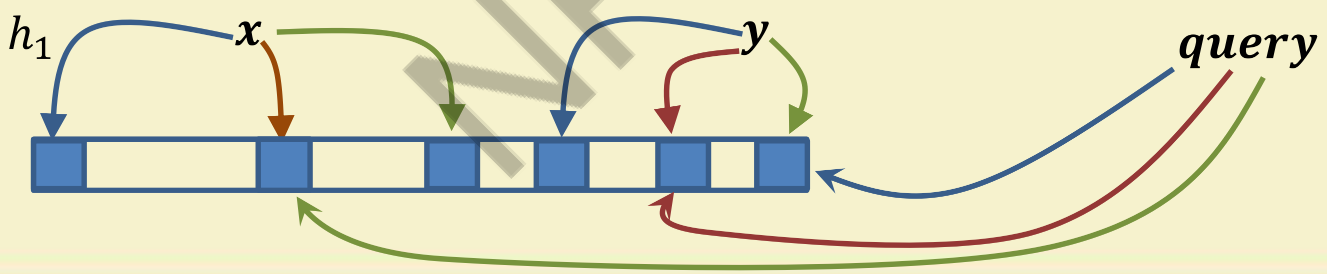
Bloom Filter

- If the element x has been added to the Bloom filter, then $Lookup(B, x)$ always return PRESENT

Bloom Filter

- If the element x has been added to the Bloom filter, then $Lookup(B, x)$ always return PRESENT

- If x has not been added to the filter before?
  - $Lookup$ sometimes still return PRESENT



Designing Bloom Filter

- Want to minimize the probability that we return a false positive
- Parameters $m = |B|$ and $k =$ number of hash functions
- $k = 1 \;\Rightarrow$ normal bit-array

- What is effect of changing k?

Effect of number of hash functions

- Increasing k
  - Possibly makes it harder for false positives to happen in $Lookup$ because of $\wedge_{i\in\{1,\ldots k\}} B[h_i(x)]$
  - But also increases the number of filled up positions
- We can analyse to find out an "optimal k"

False positive analysis

- $m = |B|$, n elements inserted
- If x has not been inserted, what is the probability that $Lookup(B, x)$ returns PRESENT?

False positive analysis

- $m = |B|$, n elements inserted
- If x has not been inserted, what is the probability that $Lookup(B, x)$ returns PRESENT?
- Assume $\{h_1, h_2, \ldots h_k\}$ are independent and $\Pr[h_i(\cdot) = j] = \frac{1}{m}$ for all positions j
- $\Pr[h_i(x) = 0] = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$

False positive analysis

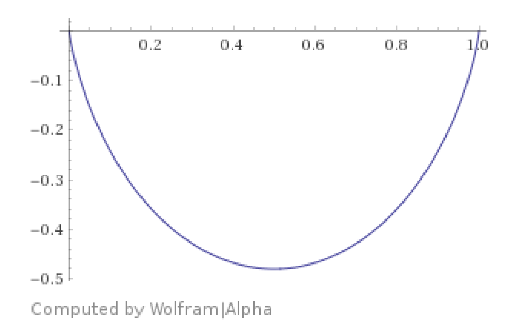
- The expected number of zero bits $\approx me^{-kn/m}$ w.h.p.
- Pr[$Lookup(B, x)$ $=$ PRESENT) $= \left(1 - e^{-kn/m}\right)^k$
- Can we choose k to minimize this probability

Choosing number of hash functions

- $p = e^{-kn/m}$
- Log (False Positive) =

$$\log(1 - p)^k = k \log(1 - p) = -\frac{m}{n} \log(p) \log(1 - p)$$

Minimized at $p = \frac{1}{2}$, i.e. $k = \mathrm{m} \log(2)/\mathrm{n}$



Bloom filter design

- This "optimal" choice gives false positive = $2^{-m \log(2)/n}$
- If we want a false positive rate of δ , set $m = \left\lceil \frac{\log\left(\frac{1}{\delta}\right) n}{\log^2(2)} \right\rceil$

<u>Example</u>: If we want 1% FPR, we need 7 hash functions and total $10n$ bits

Applications

- Widespread applications whenever small false positives are tolerable
- Used by browsers
  - to decide whether an URL is potentially malicious: a BF is used in browser, and positives are actually checked with the server.
- Databases e.g. BigTable, HBase, Cassandra, Postgrepsql use BF to avoid disk lookups for non-existent rows/columns
- Bitcoin for wallet synchronization….

Handling deletions

- Chief drawback is that BF does not allow deletions
- Counting Bloom Filter [Fan et al 00]
  - Every entry in BF is a small counter rather than a single bit
  - $Insert(x)$ increments all counters for $\{h_i(x)\}$ by 1
  - $Delete(x)$ decrements all $\{h_i(x)\}$ by 1
  - maintains 4 bits per counter
  - False negatives can happen, but only with low probability

Other Extensions

- Many recent work on Bloom filters
  - Can we do with less hashing?
  - Can BFs be compressed (needed for distributed systems)
  - Are there better structures that use less space, less randomness and less memory lookups?

References:

- Primary reference for this lecture
  - Survey on Bloom Filter, Broder and Mitzenmacher 2005, https://www.eecs.harvard.edu/~michaelm/postscripts/im2005b.pdf
  - http://www.firatatagun.com/blog/2016/09/25/bloom-filters-explanation-use-cases-and-examples/
- Others
  - Randomized Algorithms by Mitzenmacher and Upfal.

Thank You!!

IIT KHARAGPUR

IIT GANDHINAGAR

NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Scalable Data Science

Lecture 8: Streaming model, counting distinct elements

Anirban Dasgupta

Computer Science and Engineering

IIT GANDHINAGAR

IIT Gandhinagar

Indian Institute of Technology Gandhinagar

Large Data

- Data is massive, growing faster than our ability to store or index
- Predicted growth of data @ 1.7Mb/person/second [Forbes]
- Scientific data:
  - Large Hadron Collider
  - Gravitational wave detector
  - Personalized genome sequences

Handling velocity + volume

- Can we process data without explicitly storing all of it in memory? E.g. in a network switch,
  - which IPs have most packets passing through a switch
  - has traffic pattern changed overnight?

Handling velocity + volume

- Can we process data without explicitly storing all of it in memory? E.g. in a network switch,
  - which IPs have most packets passing through a switch
  - has traffic pattern changed overnight?

- We have to give up on exact answer, and rely on…
  - approximation: return answer close to truth
  - randomization: be correct only with high probability

Streaming model: sketches

- Data is assumed to come as a stream of values
  - e.g. bytes seen when reading off a tape-drive
  - destination IPs seen by a network switch
- Size of universe/stream is much large compared to available memory
  - typically assume memory is $poly(\log)$
  - Can make limited (possibly single) pass over data
  - Will create a "sketch" : a summary data structure used to answer queries at the end

Streaming problem: distinct count

- Universe is U, number of distinct elements = n, stream size is m
  - Example: $U =$ all IP addresses

    10.1.21.10, 10.93.28,1,.....,98.0.3.1,.....10.93.28.1.....

  - IPs can repeat
  - Want to estimate the number of distinct elements in the stream

Other applications

- Universe = set of all k-grams, stream is generated by document corpus
  - need number of distinct k-grams seen in corpus
- Universe = telephone call records, stream generated by tuples (caller, callee)
  - need number of phones that made > 0 calls

Solutions

- Naïve solution : $O(n\log(U))$ space
  - store all the elements, sort and count distinct
  - store a hash map, insert only if not present in map
- Bit array: $O(|U|)$ space
  - bits initialized to 1 only if element seen in stream

- Can we do this in less space? Not when exact solution needed!!

Approximations

- (ϵ, δ) −approximations
  - Algorithm will use random hash functions
  - Will return an answer $\hat{n}$ such that

$$(1-\epsilon)n \leq \hat{n} \leq (1+\epsilon)n$$

  - This will happen with probability $1-\delta$ over the randomness of the algorithm

First effort

- Stream length: m, universe size: n
- Proposed algo: Given space S, sample S items from the stream
  - Find the number of distinct elements in this set: $\hat{n}$
  - return $\hat{n} \times \frac{m}{S}$

First effort

- Stream length: m, distinct elements: n
- Proposed algo: Given space S, sample S items from the stream
  - Find the number of distinct elements in this set: $\hat{n}$
  - return $\hat{n} \times \frac{m}{S}$
- Not a constant factor approximation
  - $\underbrace{1,1,1,1,\ldots.1}_{m-n+1},2,3,4,\ldots.,n-1$

Linear Counting

- Bit array B of size m, initialized to all zero
- Hash function h: $[u] \longrightarrow [m]$
- When seeing item x , set $B[h(x)] = 1$

Linear Counting

- Bit array B of size m, initialized to all zero
- Hash function h: $[n] \rightarrow [m]$
- When seeing item x , set $B[h(x)] = 1$

- $z_m =$ Number of zero entries
- Return estimate $-\mathrm{m}\log(\frac{z_m}{m})$

Linear Counting Analysis

- Pr[position remaining 0] $= \left(1 - \frac{1}{m}\right)^n \approx e^{-\frac{n}{m}}$
- Expected number of positions at zero = $\mathrm{E}[\mathrm{z_m}] = me^{-n/m}$

- Using tail inequalities we can show this is concentrated
- Typically useful only for $m = \Theta(n)$, often useful in practice

Flajolet Martin Sketch

- Components
  - “random” hash function $h: U \to 2^{\ell}$ for some large ℓ
  - $h(x)$ is a ℓ −length bit string
  - initially assume it is completely random, can relax
- $zero(v) =$ position of rightmost 1 in bit representation of v

$$= \max\{\, i\,, 2^{i} \ divides\, v\,\}$$

  - $zeros(10110) = 1, \quad zeros(110101000) = 3$

Flajolet Martin Sketch

<u>Initialize:</u>

- Choose a “random” hash function $h: U \to 2^{\ell}$
- $z \leftarrow 0$

<u>Process(x)</u>

- if $zeros(h(x)) > z, \;\; \mathrm{z} \leftarrow zeros(h(x))$

<u>Estimate:</u>

- return $2^{z+1/2}$

Example

	h(.)
	0110101
	1011010
	1000100
	1111010

Space usage

- We need $\ell \geq C\log(n)$ for some $C \geq 3$, say
  - by birthday paradox analysis, no collisions with high prob

- Sketch : z , needs to have only $O(\log\log n)$ bits !!!
- Total space usage = $O(\log n + \log\log n)$

Intuition

- Assume hash values are uniformly distributed
- The probability that a uniform bit-string
  - is divisible by 2 is ½
  - is divisible by 4 is ¼
  - ….
  - is divisible by 2^k is $\frac{1}{2^k}$
- We don't expect any of them to be divisible by $2^{\log_2(n)+1}$

Formalizing intuition

- S = set of elements that appeared in stream
- For any $r \in [\ell], j \in S$, X_{rj} = indicator of $\mathrm{zeros}(h(j)) \geq r$
- Y_r = number of $j \in S$ such that $\mathrm{zeros}(h(j)) \geq r$

$$Y_r = \sum_{j \in S} X_{rj}$$

- Let $\hat{z}$ be final value of z after algo has seen all data

Proof of FM

- $Y_r > 0 \leftrightarrow \hat{z} \geq r$, equivalently, $Y_r = 0 \leftrightarrow \hat{z} < r$

Proof of FM

- $Y_r > 0 \leftrightarrow \hat{z} \geq r$, equivalently, $Y_r = 0 \leftrightarrow \hat{z} < r$

- $E[Y_r] = \sum_{j \in S} E\left[X_{rj}\right]$ $\quad X_{rj} = \begin{cases} 1 \text{ with prob } \frac{1}{2^r} \\ 0 \quad \text{else} \end{cases}$

- $E[Y_r] = \frac{n}{2^r}$ $\quad var(Y_r) = \sum_{j \in S} var(X_{rj}) \leq \sum_{j \in S} E\left[X_{rj}^2\right]$

Proof of FM

- $var(Y_r) \leq \sum_{j \in S} E\left[X_{rj}^2\right] \leq \; n/2^r$

$$\Pr[Y_r > 0] = \Pr[Y_r \geq 1] \leq \frac{E[Y_r]}{1} = \frac{n}{2^r}$$

Proof of FM

- $var(Y_r) \leq \sum_{j \in S} E\left[X_{rj}^2\right] \leq \; n/2^r$

$$\Pr[Y_r > 0] = \Pr[Y_r \geq 1] \leq \frac{E[Y_r]}{1} = \frac{n}{2^r}$$

$$\Pr[Y_r = 0] \leq \Pr\left[\,|Y_r - E[Y_r]| \geq E[Y_r]\right] \leq \frac{var(Y_r)}{E[Y_r]^2} \leq \frac{2^r}{n}$$

Upper bound

Returned estimate $\hat{n} = 2^{\hat{z}+1/2}$

a = smallest integer with $2^{a+1/2} \geq 4n$

$$\Pr[\hat{n} \geq 4n] = \Pr[\hat{z} \geq a] = \Pr[Y_a > 0] \leq \frac{n}{2^a} \leq \frac{\sqrt{2}}{4}$$

Lower bound

Returned estimate $\hat{n} = 2^{\hat{z}+1/2}$

b = largest integer with $2^{b+1/2} \leq n/4$

$$\Pr\left[\hat{n} \leq \frac{n}{4}\right] = \Pr[\,\hat{z} \leq b\,] = \Pr[Y_{b+1} = 0] \leq \frac{2^{b+1}}{n} \leq \frac{\sqrt{2}}{4}$$

Understanding the bound

- By union bound, with prob $1 - \frac{\sqrt{2}}{2}$,

$$\frac{n}{4} \leq \hat{n} \leq 4n$$

- Can get somewhat better constants
- Need only 2-wise independent hash functions, since we only used variances

Improving the probabilities

- To improve the probabilities, a common trick: median of estimates
- Create $\widehat{z_1}, \widehat{z_2}, \ldots., \widehat{z_k}$ in parallel
  - return median
- Expect at most $\frac{\sqrt{2}}{4}$ of them to exceed $4n$

Improving the probabilities

- To improve the probabilities, a common trick: median of estimates
- Create $\widehat{z_1}, \widehat{z_2}, \ldots, \widehat{z_k}$ in parallel
  - return median
- Expect at most $\frac{\sqrt{2}}{4}k$ of them to exceed $4n$
- But if median exceeds $4n$, then $\frac{k}{2}$ of them does → using Chernoff bound this prob is $\exp(-\Omega(k))$

Improving the probabilities

- To improve the probabilities, a common trick: median of estimates
- Create $\widehat{z_1}, \widehat{z_2}, \ldots, \widehat{z_k}$ in parallel
  - return median
- Using Chernoff bound, can show that median will lie in $\left[\frac{n}{4}, 4n\right]$ with probability $1 - \exp(-\Omega(k))$.
- Given error prob δ, choose $k = O(\log\left(\frac{1}{\delta}\right))$

Summary

- Streaming model– useful abstraction
  - Estimating basic statistics also nontrivial

- Estimating number of distinct elements
  - Linear counting
  - Flajolet Martin

k-MV sketch

- Developed in an effort to get better accuracy

- Additional capabilities for estimating cardinalities of union and intersection of streams
  - If S_1 and S_2 are two streams, can compute their union sketch from individual sketches of S_1 and S_2

[kMV sketch slides courtesy Cohen-Wang]

Sampling via hashing: Thought experiment

- Suppose $h: U \to [0,1]$ is random hash function such that $h(x) \sim U[0,1]$ for all $x \in U$

- Maintain min-hash value y
  - initialize $y \leftarrow 1$
  - For each item x_i, $y \leftarrow \min(y, h(x_i))$

[kMV sketch slides courtesy Cohen-Wang]

Example

	h(.)
	0110101
	1011010
	1000100
	1111010

Intuition

- What information does y have about the number of distinct elements n ?

- Expectation of minimum is $E[\min_i h(x_i)] = \frac{1}{n+1}$

Why is expectation of min = $\frac{1}{n+1}$?

- Imagine a circle instead of $[0, 1]$
- Choose $n + 1$ points uniformly at random

Why is expectation of min = $\frac{1}{n+1}$?

- Imagine a circle instead of $[0, 1]$
- Choose $n + 1$ points uniformly at random
- $n + 1$ intervals are formed
- Expected length of each interval is $\frac{1}{n+1}$

Why is expectation of min = $\frac{1}{n+1}$?

- Imagine a circle instead of $[0, 1]$
- Choose $n + 1$ points uniformly at random
- $n + 1$ intervals are formed
- Expected length of each interval is $\frac{1}{n+1}$
- Think of the first point as the place to cut the circle!

[kMV sketch slides courtesy Cohen-Wang]

k-minimum value sketch

<u>Initialize:</u>

– $y_1, \ldots y_k \leftarrow 1, \ldots 1$

<u>Process(x):</u>

– For all $j \in [k]$, $y_j \leftarrow \min(y_j, h(x_i))$

<u>Estimate:</u>

– return median-of-means$(\frac{1}{y_1}, \ldots, \frac{1}{y_k})$

Median-of-means

- Given (ϵ, δ) , choose $k = \frac{c}{\epsilon^2} \log(\frac{1}{\delta})$
- Group $t_1, \ldots t_k$ into $\log(\frac{1}{\delta})$ groups of size $\frac{c}{\epsilon^2}$ each
- Find $\text{mean}(t_i)$ for each group: $Z_1, \ldots Z_{\log(\frac{1}{\delta})}$
- Return $\hat{n} = \text{median of } Z_1, \ldots Z_{\log(\frac{1}{\delta})}$

Example

	h1	h2	h3	h4
	.45	.19	.10	.92
	.35	.51	.71	.20
	.21	.07	.93	.18
	.14	.70	.50	.25

Complexity

- Total space required = $O(k \log n) = O(\frac{1}{\epsilon^2} \log n \log(\frac{1}{\delta}))$
  - can be improved
  - don't need floating points, can use $h: U \to 2^{\ell}$ as before
  - can do with k-wise universal hash functions

- Update time per item = $O(k)$
  - However, can show that most items will not result in updates

Theoretical Guarantees

With probability $1 - \delta$, returns $\hat{n}$ satisfies

$$(1 - \epsilon)n \leq \hat{n} \leq (1 + \epsilon)n$$

Proof is simple application of expectation and Chernoff bound

Merging

- For two stream S_1 and S_2 use same set of hash functions
- For each j $\in [k]$, $\text{find}\ \min(y_j, y_j')$
- Gives estimate of $|S_1 \cup S_2|$

References:

- Primary reference for this lecture
  - Lecture notes by Amit Chakrabarti: http://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf

- Others
  - Blum, Hopcroft, Kannan.
  - Sketch techniques for approximate query processing, Graham Cormode. http://dimacs.rutgers.edu/~graham/pubs/papers/sk.pdf

Thank You!!

IIT KHARAGPUR

IIT GANDHINAGAR

NPTEL ONLINE
CERTIFICATION COURSES

NPTEL

Scalable Data Science

Lecture 9: Frequent Elements

Anirban Dasgupta

Computer Science and Engineering

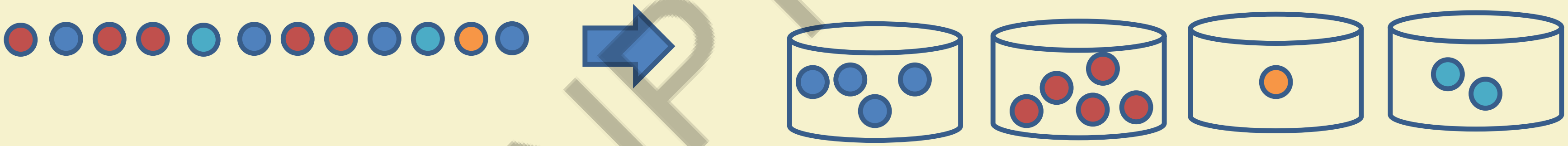
IIT GANDHINAGAR

Streaming model revisited

- Data is seen as incoming sequence
  - can be just element-ids, or ids +frequency updates
- Arrival only streams
- Arrival + departure
  - Negative updates to frequencies possible
  - Can represent fluctuating quantities, e.g.

Frequency Estimation

- Given the input stream, answer queries about item frequencies at the end
  - Useful in many practical applications e.g. finding most popular pages from website logs, detecting DoD attacks, database optimization



- Also used as subroutine in many problems
  - Entropy estimation, itemset mining etc

[Slides courtesy of Graham Cormode]

Frequency estimation

Q1. Can we create a data structure, sketch, sublinear in the data size to answer all frequency queries accurately?

Frequency estimation in one pass

Q1. Can we create a data structure, sketch, sublinear in the data size to answer all frequency queries accurately?

– No

Q2. Can we create a sketch to estimate frequencies of the "most frequent" elements exactly?

Frequency estimation in one pass

Q1. Can we create a data structure, sketch, sublinear in the data size to answer all frequency queries exactly?

- No

Q2. Can we create a sketch to answer frequencies of the “most frequent” elements exactly?

- No

Q3. Sketch to estimate frequencies of “most frequent” elements approximately?

Frequency estimation in one pass

Q1. Can we create a data structure, sketch, sublinear in the data size to answer all frequency queries exactly?

- No

Q2. Can we create a sketch to answer frequencies of the “most frequent” elements exactly?

- No

Q3. Sketch to estimate frequencies of “most frequent” elements approximately?

- YES!

Approximate Heavy Hitters

- Given an update stream of length m, find out all elements that occur "frequently"
  - e.g. at least 1% of the time
  - cannot be done in sublinear space, one pass
- Find out elements that occur at least ϕm times, and none that appears $< (\phi - \epsilon)m$ times
  - Error ϵ
  - Related question: estimate each frequency with error $\pm\epsilon m$

Starting with a puzzle

[J. Algorithms, 1981] Suppose we have a list of N numbers, representing votes of N processors on result of some computation. We wish to decide if there is a majority vote and what that vote is.

- By J.S. Moore
- Did not talk about streaming solution, but proposed solution is
- Strict majority: >N/2

Majority Algorithm

- Arrivals only model
- Start with a counter set to zero
- For each item
  - if counter = 0, pick new item and increment counter
  - else if new item is same as item in hand, increment counter
  - else decrement counter

Majority Algorithm

- Start with a counter set to zero
- For each item
  - if counter = 0, pick new item and increment counter
  - else if new item is same as item in hand, increment counter
  - else decrement counter
- If there is a majority item, it is in hand at the end
- Proof: Since majority occurs > N/2 times, not all occurrences can be cancelled out

Frequent [Misra-Gries]

- Keep k counters and items in hand

Initialize:

– Set all counters to 0

Process(x)

– if x is same as any item in hand, increment its counter

– else if number of items $< k$, store x with counter $= 1$

– else drop x and decrement all counters

Query(q)

– If q is in hand return its counter, else 0

Frequent

- f_x be the true frequency of element x
- At the end, some set of elements is stored with counter values
- If $query\ y$ in hand, $\widehat{f_y}$ = counter value, else $\widehat{f_y} = 0$

Example

Theoretical Bound

Claim: No element with frequency $> m/k$ is missed at the end

Theoretical Bound

Claim: No element with frequency $> m/k$ is missed at the end

Intuition: Each decrement (including drop) is charged with k arrivals. Therefore, will have some copy of an item with frequency $> m/k$

Stronger Claim

Choose $k = \frac{1}{\epsilon}$. For every item x, with frequency f_x the algo can return an estimate $\widehat{f_x}$ such that

$$f_x - \epsilon m \leq \widehat{f_x} \leq f_x$$

Stronger Claim

Choose $k = \frac{1}{\epsilon}$. For every item x, with frequency f_x the algo can return an estimate $\widehat{f_x}$ such that

$$f_x - \epsilon m \le \widehat{f_x} \le f_x$$

Same intuition, whenever we drop a copy of item x, we also drop $k - 1$ copies of other items

Summary

- Simple deterministic algorithm to estimate heavy hitters
  - Works only in the arrival model
- Proposed in 1982, rediscovered multiple times with modifications
- Also basis of matrix low rank approximation
- Our next lecture will discuss other algorithms

References:

- Primary references for this lecture
  - Lecture slides by Graham Cormode
    http://dmac.rutgers.edu/Workshops/WGUnifyingTheory/Slides/cormode.pdf
  - Lecture notes by Amit Chakrabarti: http://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf
  - Sketch techniques for approximate query processing, Graham Cormode. http://dimacs.rutgers.edu/~graham/pubs/papers/sk.pdf

Thank You!!