Lecture 7: Probabilistic Data Structure

Xiaotie Deng

AIMS Lab
Department of Computer Science
Shanghai Jiaotong University

October 23, 2017

Probability Data Structure

Overview

- Bloom Filter: Membership Test
- Merkle Tree:
- HyperLogLog Algorithm

A Simple Bloom Filter

- Membership test returns
  - possibly in set
  - definitely not in set
- Bit Array of size m bits using hasing
  - hash function h: $h(S) \in [0, m)$.
  - Insert s : $A(h(s)) = 1$.
  - Query s : return $A(h(s))$.
- One hashing of n items in a bit array of size m.
  - False-positive rate: $1 - (1 - \frac{1}{m})^n = 1 - e^{-n/m}$
  - It is roughly 10.0% if $n = 0.1m$.
  - Comment: My latex calculator has a large error rate.

Performance Analysis

- Use k hashing functions simultaneously on same hash array.
  - A member a if $h_1(a) = h_2(a) = \cdots = h_k(a) = 1$.
  - False-positive rate:
    $[1 - (1 - \frac{1}{m})^{kn}]^k = (1 - e^{-kn/m})^k = \exp(k \ln(1 - e^{-kn/m}))$
- Optimal k: $\ln(1 - e^{-kn/m}) + \frac{k}{1-e^{-kn/m}} * (-\frac{n}{m} e^{-kn/m}) = 0$
  - $\ln(1 - e^{-t}) - \frac{t}{1-e^{-t}} * e^{-t} = 0$
  - $t = kn/m = \ln 2$, $k_{opt} = \frac{m}{n} \ln 2$, & the false-positive rate
  - $[1 - (1 - \frac{1}{m})^{kn}]^k = (1 - e^{-kn/m})^k = \exp(k \ln(1 - e^{-kn/m}))$
- False positive rate is 0.81787% for $n/m = 0.1 \Rightarrow k_{opt} = 7$.
- Exercise: How full is the hash array?

Extension

- Deletion cannot be done by setting a location to zero as it may correspond to several hash locations.
- Related Works
  - Tarkoma, Sasu, Christian Esteve Rothenberg, and Eemil Lagerspetz. "Theory and practice of bloom filters for distributed systems." IEEE Communications Surveys & Tutorials 14.1 (2012): 131-155.
  - Goodrich, Michael T., and Michael Mitzenmacher. "Invertible bloom lookup tables." Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on. IEEE, 2011.
  - Chen, Ruichuan, et al. "Towards statistical queries over distributed private user data." Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012.

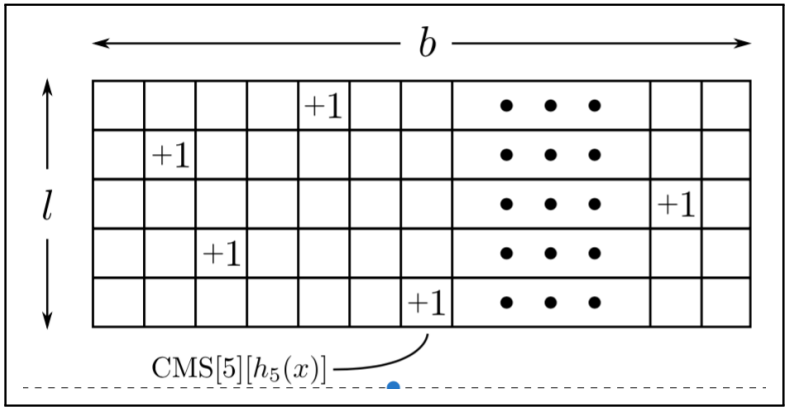
Count-Min Sketching

Overview

- An $l \times b$ array to support two operations: Inc(x) and Count(x)
  - Return frequency of x.
  - Inc(x): invoked each time when x is entered into the database.
- Two parameters: b the bucket # (2000), l (5) the number of hash functions.
  - b to compress the array A: $b << n$, may create errors.
  - implement independent trials to reduce errors.

Operation Increment

- Initiate CMS to all zeros.
- for i=1 to l increment $CMS[i][h_i(x)]$.
- Running time $O(l)$.



Operation Count

- Return $\min_{i=1}^{l}\{CMS[i][h_i(x)]\}$.
- Running time $O(l)$.
- $CMS[i][h_i(x)] \geq f(x)$, where $f(x)$ the frequency of item x entered into the database.
- How large is the over-estimate?

Error Estimation

- $f(x)$ true count, Z_i over-estimation
- $Z_i = f(x) + \sum_{y \in S} f_y$, where $S = \{y \neq x : h_i(x) = h_i(y)\}$
- Random hashing: $Pr[h_i(x) = h_i(y) : x \neq y] = \frac{1}{b}$.
- $E[Z_i] = f_x + \sum_{y \neq x} f_y E[1_{h(y)=h(x)}] \leq f_x + \frac{n}{b}$.

Error in High Probability

- $f(x)$ true count, Z_i over-estimation
- $Z_i = f(x) + \sum_{y \in S} f_y$, where $S = \{y \neq x : h_i(x) = h_i(y)\}$
  - Random hashing: $Pr[h_i(x) = h_i(y) : x \neq y] = \frac{1}{b}$.
  - $E[Z_i] = f_x + \sum_{y \neq x} f_y E[1_{h(y)=h(x)}] \leq f_x + \frac{n}{b}$.
- Define $X = Z_i - f(x)$ be the overestimate on the i-th row.
  - choose $b = \frac{e}{\epsilon} \implies E[X] \leq \frac{\epsilon n}{e}$.
  - $Pr[Z_i \geq f_x + \epsilon n] = Pr[X \geq e\frac{\epsilon n}{e}] \leq \frac{1}{e}$.
- Count(x) exceeds f_x by more than ϵn only if every row is estimated too bigger: $Pr[\min_{i=1}^{l} Z_i \geq f_x + \epsilon n] \leq \frac{1}{\epsilon^l}$.

Hashing Functions

Overview

- Determinism: The same input should have the same hash value.
- Uniformity: map inputs as evenly as possible over the output space.
- Perfect hashing: Map set S to different elements.
- Universal hashing: given any two inputs, randomly pick a hash function from this class, the two inputs are hashed into the same output with a probability no more than $1/m$.

Random Oracle

- A theoretical black box that responds to a query
  - for the first time: return a random number uniformly from its output domain
  - for the subsequent times: the same as the first time return.
- Random oracle security concept: A system that is proven secure if every hash function is replaced by a random oracle.
- Random oracle hypothesis (proven false): Complexity classes separable with probability 1 implies deterministic separability.
  - Counter example: IP=PSPACE

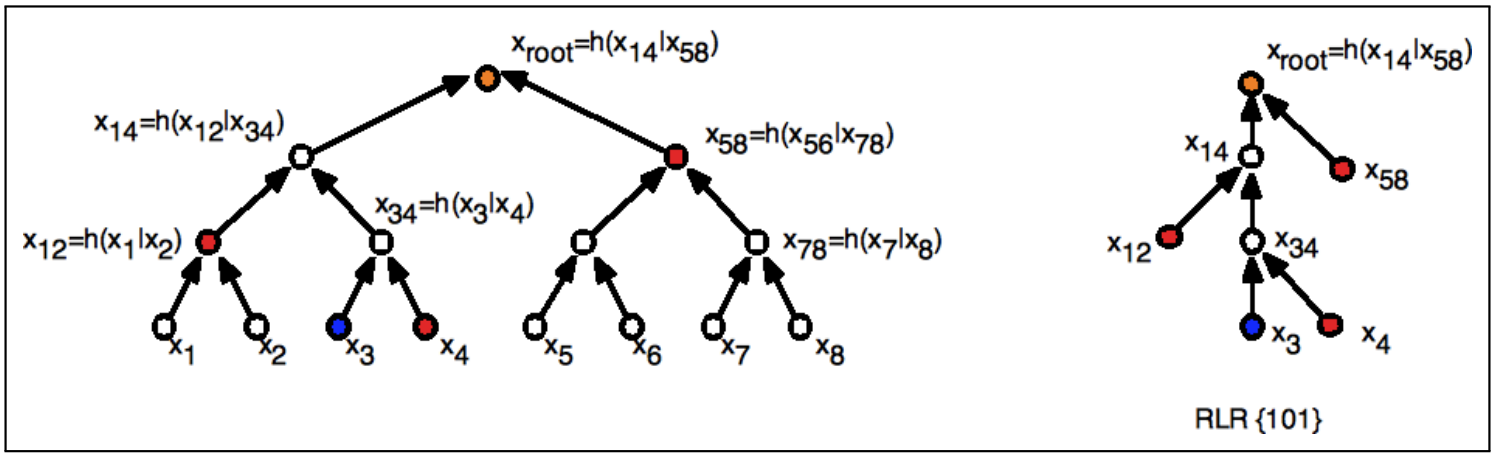
REFERENCE: https://en.wikipedia.org/wiki/Random_oracle

Various Desirable Properties

- Continuity: Similar data with similar hash values.
- Order preserving: $a < b$ iff $h(a) < h(b)$.
- One way hashing: not able to find input from output.

Hashing Chain

- $h^1(x)h(x)$, $h^{i+1}(x) = h^i(x), \ldots$
- password protection
- hyperledger of data sequence for protection of integrity.



By guardtime.com - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=23758966

Password Chain

- Generate a chain of $h^1(x)$, $h^2(x)$, $\cdots$, $h^n(x)$.
- Send $t = h^n(x)$ to system as one's password with one way hash function h.
- When login to system, submit $p = h^{n-1}$.
- System admits the user if $h(p) = t$ and replace $t \leftarrow p$.