IIT KHARAGPUR

IIT GANDHINAGAR

NPTEL ONLINE
CERTIFICATION COURSES

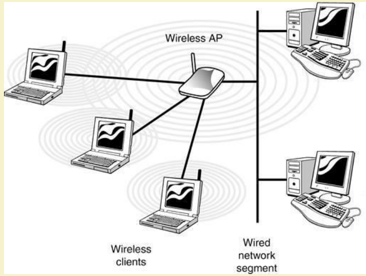
Scalable Data Science

Lecture 7: Bloom Filters

Anirban Dasgupta
Computer Science and Engineering
IIT 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

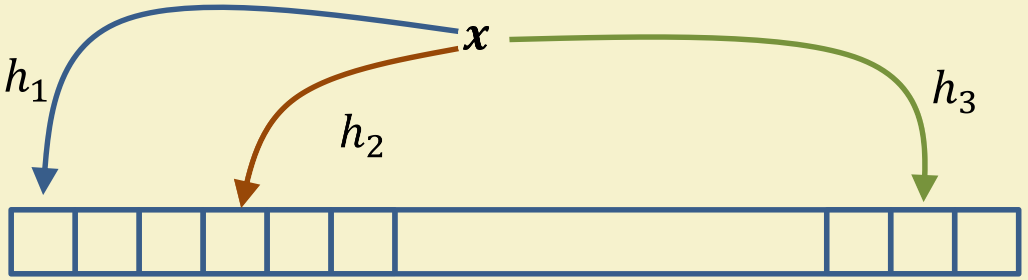
- In hash table construction, we used random hash functions
  - we never return incorrect answer
  - query time is a random variable
  - These are Las Vegas algorithms

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

Bloom filter

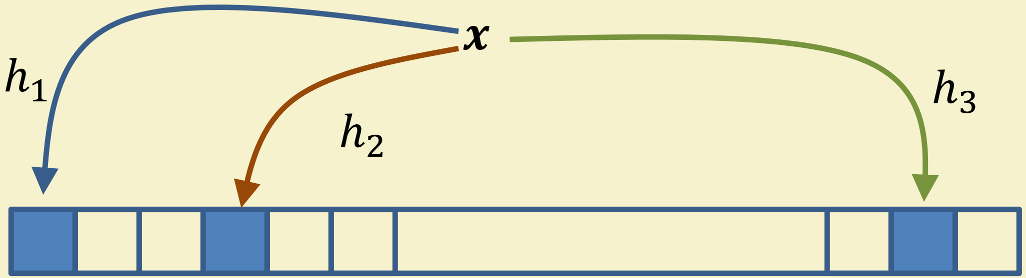
[Bloom, 1970]

- A bit-array B, $|B| = m$
- k hash functions, $h_1, h_2, \ldots, h_k$, 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 $\wedge_{i \in \{1,\ldots k\}} B[h_i(x)]$, return PRESENT, else ABSENT

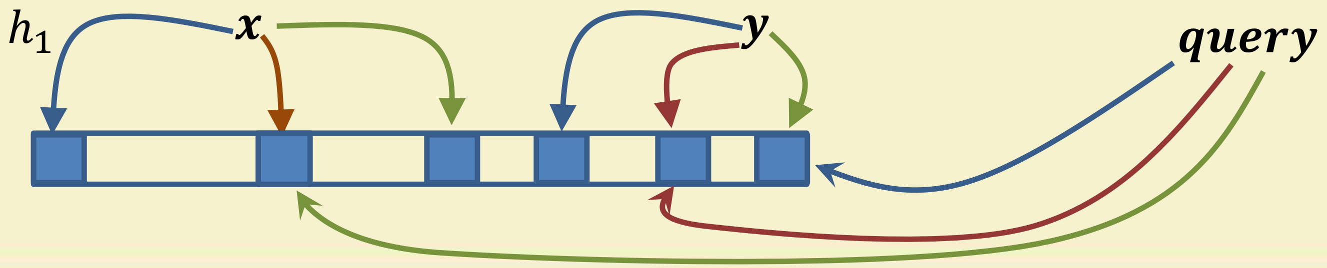
Bloom Filter

- 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

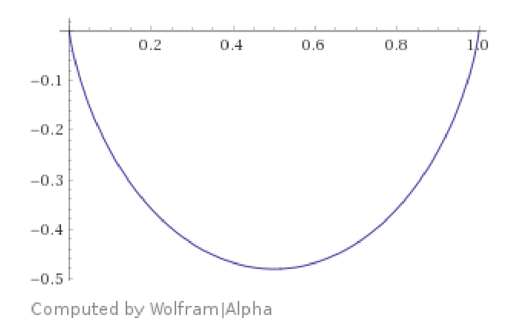
- The expected number of zero bits $\approx me^{-kn/m}$ w.h.p.
- Pr[$Lookup(B, x)$ $=$ PRESENT) $= \left(1 - e^{-kn/m}\right)^k$
- Can we choose k to minimize this probability

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 $m = \left\lceil \frac{\log\left(\frac{1}{\delta}\right) n}{\log^2(2)} \right\rceil$

Example: 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!!