COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2022.

Lecture 6

Logistics

- Problem Set 1 is due tomorrow at 11:59pm in Gradescope.
- Quiz 3 is due Monday at 8pm.

Last Time

Last Class:

- Higher moment bounds and exponential concentration bounds
- Bernstein inequality

This Class:

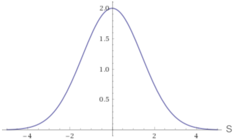
- Connection between exponential concentration bounds and the central limit theorem.
- The Chernoff bound.
- Bloom filters: random hashing to maintain a large set in small space.

Interpretation as a Central Limit Theorem

Bernstein Inequality (Simplified): Consider independent random variables $X_1, \ldots, X_n$ falling in [-1,1]. Let $\mu = \mathbb{E}[\sum X_i]$, $\sigma^2 = \text{Var}[\sum X_i]$, and $s \leq \sigma$. Then:

$$\Pr\left(\left|\sum_{i=1}^{n} X_i - \mu\right| \geq s\sigma\right) \leq 2\exp\left(-\frac{s^2}{4}\right).$$

Can plot this bound for different s:

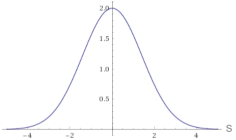


Interpretation as a Central Limit Theorem

Bernstein Inequality (Simplified): Consider independent random variables $X_1, \ldots, X_n$ falling in [-1,1]. Let $\mu = \mathbb{E}[\sum X_i]$, $\sigma^2 = \text{Var}[\sum X_i]$, and $s \leq \sigma$. Then:

$$\Pr\left(\left|\sum_{i=1}^{n} X_i - \mu\right| \geq s\sigma\right) \leq 2\exp\left(-\frac{s^2}{4}\right).$$

Can plot this bound for different s:



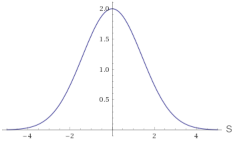
Looks a lot like a Gaussian (normal) distribution.

Interpretation as a Central Limit Theorem

Bernstein Inequality (Simplified): Consider independent random variables $X_1, \ldots, X_n$ falling in [-1,1]. Let $\mu = \mathbb{E}[\sum X_i]$, $\sigma^2 = \text{Var}[\sum X_i]$, and $s \leq \sigma$. Then:

$$\Pr\left(\left|\sum_{i=1}^{n} X_i - \mu\right| \geq s\sigma\right) \leq 2\exp\left(-\frac{s^2}{4}\right).$$

Can plot this bound for different s:



Looks a lot like a Gaussian (normal) distribution.

$$\mathcal{N}(0, \sigma^2) \text{ has density } p(s\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{s^2}{2}}.$$

Gaussian Tails

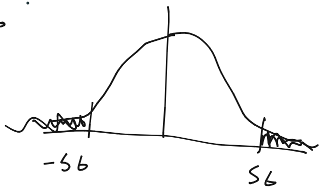
$\mathcal{N}(0, \sigma^2)$ has density $p(s\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{s^2}{2}}$.

Gaussian Tails

$$\mathcal{N}(0, \sigma^2) \text{ has density } p(s\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{s^2}{2}}.$$

Exercise: Using this can show that for $X \sim \mathcal{N}(0, \sigma^2)$: for any $s \geq 0$,

$$\Pr\left(|X| \geq s \cdot \sigma\right) \leq 2e^{-\frac{s^2}{2}}.$$



Gaussian Tails

$$\mathcal{N}(0, \sigma^2) \text{ has density } p(s\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{s^2}{2}}.$$

Exercise: Using this can show that for $X \sim \mathcal{N}(0, \sigma^2)$: for any $s \geq 0$,

$$\Pr(|X| \geq s \cdot \sigma) \leq 2e^{-\frac{s^2}{2}}.$$

$2e^{-s^2/4}$

Essentially the same bound that Bernstein's inequality gives!

Gaussian Tails

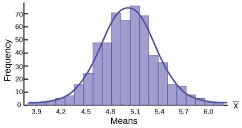
$$\mathcal{N}(0, \sigma^2) \text{ has density } p(s\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{s^2}{2}}.$$

Exercise: Using this can show that for $X \sim \mathcal{N}(0, \sigma^2)$: for any $s \geq 0$,

$$\Pr\left(|X| \geq s \cdot \sigma\right) \leq 2e^{-\frac{s^2}{2}}.$$

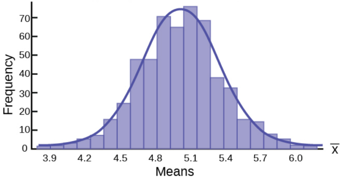
Essentially the same bound that Bernstein's inequality gives!

Central Limit Theorem Interpretation: Bernstein's inequality gives a quantitative version of the CLT. The distribution of the sum of *bounded* independent random variables can be upper bounded with a Gaussian (normal) distribution.



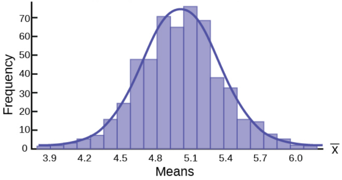
Central Limit Theorem

Stronger Central Limit Theorem: The distribution of the sum of *n* *bounded* independent random variables converges to a Gaussian (normal) distribution as *n* goes to infinity.



Central Limit Theorem

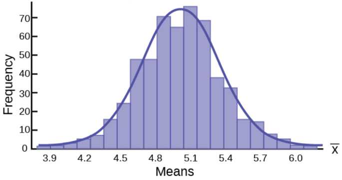
Stronger Central Limit Theorem: The distribution of the sum of n *bounded* independent random variables converges to a Gaussian (normal) distribution as n goes to infinity.



- Why is the Gaussian distribution is so important in statistics, science, ML, etc.?

Central Limit Theorem

Stronger Central Limit Theorem: The distribution of the sum of *n* *bounded* independent random variables converges to a Gaussian (normal) distribution as *n* goes to infinity.



- Why is the Gaussian distribution is so important in statistics, science, ML, etc.?
- Many random variables can be approximated as the sum of a large number of small and roughly independent random effects. Thus, their distribution looks Gaussian by CLT.

A useful variation of the Bernstein inequality for binary (indicator) random variables is:

Chernoff Bound (simplified version): Consider independent random variables $X_1, \ldots, X_n$ taking values in $\{0, 1\}$. Let $\mu = \mathbb{E}[\sum_{i=1}^n X_i]$. For any $\delta \geq 0$

$\sum \mathbb{E} X_i$

$$\Pr\left(\left|\sum_{i=1}^n X_i - \mu\right| \geq \delta\mu\right) \leq 2\exp\left(-\frac{\delta^2\mu}{2+\delta}\right).$$

The Chernoff Bound

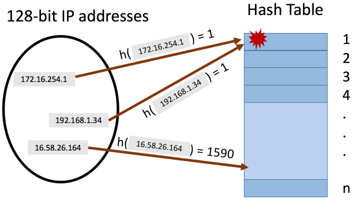
A useful variation of the Bernstein inequality for binary (indicator) random variables is:

Chernoff Bound (simplified version): Consider independent random variables $X_1, \ldots, X_n$ taking values in $\{0, 1\}$. Let $\mu = \mathbb{E}[\sum_{i=1}^{n} X_i]$. For any $\delta \geq 0$

$$\Pr\left(\left|\sum_{i=1}^{n} X_i - \mu\right| \geq \delta\mu\right) \leq 2\exp\left(-\frac{\delta^2 \mu}{2 + \delta}\right).$$

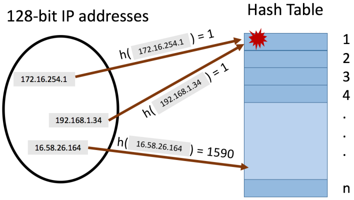
As δ gets larger and larger, the bound falls of exponentially fast.

Return to Random Hashing



We hash m values $x_1, \ldots, x_m$ using a random hash function into a table with $n = m$ entries.

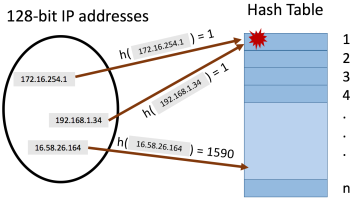
Return to Random Hashing



We hash m values $x_1, \ldots, x_m$ using a random hash function into a table with $n = m$ entries.

- I.e., for all $j \in [m]$ and $i \in [m]$, $\Pr(\mathbf{h}(x_j) = i) = \frac{1}{m}$ and hash values are chosen independently. fully independent

Return to Random Hashing



We hash m values $x_1, \ldots, x_m$ using a random hash function into a table with $n = m$ entries.

- I.e., for all $j \in [m]$ and $i \in [m]$, $\Pr(\mathbf{h}(x_j) = i) = \frac{1}{m}$ and hash values are chosen independently.

What will be the maximum number of items hashed into the same location?

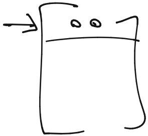
Maximum Load in Randomized Hashing

Let S_i be the number of items hashed into position i and $\mathsf{S}_{i,j}$ be 1 if x_j is hashed into bucket i ($\mathsf{h}(x_j) = i$) and 0 otherwise.

m: total number of items hashed and size of hash table. $x_1, \ldots, x_m$: the items.
h: random hash function mapping $x_1, \ldots, x_m \rightarrow [m]$.

Maximum Load in Randomized Hashing

Let S_i be the number of items hashed into position i and $\mathsf{S}_{i,j}$ be 1 if x_j is hashed into bucket i ($\mathsf{h}(x_j) = i$) and 0 otherwise.



$$\mathbb{E}[\mathsf{S}_i] = \sum_{j=1}^{m} \mathbb{E}[\mathsf{S}_{i,j}] = m \cdot \frac{1}{m} = 1$$

m: total number of items hashed and size of hash table. $x_1, \ldots, x_m$: the items.
h: random hash function mapping $x_1, \ldots, x_m \to [m]$.

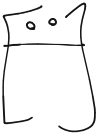
Maximum Load in Randomized Hashing

Let S_i be the number of items hashed into position i and $\mathsf{S}_{i,j}$ be 1 if x_j is hashed into bucket i ($\mathsf{h}(x_j) = i$) and 0 otherwise.

$$\mathbb{E}[\mathsf{S}_i] = \sum_{j=1}^{m} \mathbb{E}[\mathsf{S}_{i,j}] = m \cdot \frac{1}{m} = 1 = \mu.$$

m: total number of items hashed and size of hash table. $x_1, \ldots, x_m$: the items.
h: random hash function mapping $x_1, \ldots, x_m \rightarrow [m]$.

Let S_i be the number of items hashed into position i and $\mathsf{S}_{i,j}$ be 1 if x_j is hashed into bucket i ($\mathsf{h}(x_j) = i$) and 0 otherwise.



$$\mathbb{E}[\mathsf{S}_i] = \sum_{j=1}^{m} \mathbb{E}[\mathsf{S}_{i,j}] = m \cdot \frac{1}{m} = 1 = \mu.$$

By the Chernoff Bound: for any $\delta \geq 0$,

$$\Pr(\mathsf{S}_i \geq 1 + \delta) \leq \Pr\left(\left|\sum_{i=1} \mathsf{S}_{i,j} - 1\right| \geq \delta \cdot \mu\right) \leq 2\exp\left(-\frac{\delta^2}{2+\delta}\right)$$

> m: total number of items hashed and size of hash table. $x_1, \ldots, x_m$: the items.
> h: random hash function mapping $x_1, \ldots, x_m \to [m]$.

Maximum Load in Randomized Hashing

$$\Pr(\mathsf{S}_i \geq 1 + \delta) \leq \Pr\left(\left|\sum_{i=1}^{n} \mathsf{S}_{i,j} - 1\right| \geq \delta\right) \leq 2\exp\left(-\frac{\delta^2}{2+\delta}\right).$$

m: total number of items hashed and size of hash table. S_i: number of items hashed to bucket i. $\mathsf{S}_{i,j}$: indicator if x_j is hashed to bucket i. δ: any value ≥ 0.

Maximum Load in Randomized Hashing

$\geq 20\log m + 1 \approx 20\log m$

$$\Pr(S_i \geq 1+\delta) \leq \Pr\left(\left|\sum_{i=1}^{n} S_{i,j} - 1\right| \geq \delta\right) \leq 2\exp\left(-\frac{\delta^2}{2+\delta}\right).$$

Set $\delta = 20\log m$. Gives:

m: total number of items hashed and size of hash table. S_i: number of items hashed to bucket i. $S_{i,j}$: indicator if x_j is hashed to bucket i. δ: any value ≥ 0.

Maximum Load in Randomized Hashing

$$\Pr(S_i \geq 1 + \delta) \leq \Pr\left(\left|\sum_{i=1}^{n} S_{i,j} - 1\right| \geq \delta\right) \leq 2\exp\left(-\frac{\delta^2}{2+\delta}\right).$$

Set $\delta = 20 \log m$. Gives:

$$\Pr(S_i \geq 20 \log m + 1) \leq 2\exp\left(-\frac{(20 \log m)^2}{2 + 20 \log m}\right) \qquad .$$

m: total number of items hashed and size of hash table. S_i: number of items hashed to bucket i. $S_{i,j}$: indicator if x_j is hashed to bucket i. δ: any value ≥ 0.

$$\Pr(\mathsf{S}_i \geq 1+\delta) \leq \Pr\left(\left|\sum_{i=1}^{n} \mathsf{S}_{i,j} - 1\right| \geq \delta\right) \leq 2\exp\left(-\frac{\delta^2}{2+\delta}\right).$$

Set $\delta = 20\log m$. Gives:

$$\Pr(\mathsf{S}_i \geq 20\log m + 1) \leq 2\exp\left(-\frac{(20\log m)^2}{2+20\log m}\right) \leq 2\exp(-18\log m) \leq \frac{2}{m^{18}}.$$

m: total number of items hashed and size of hash table. S_i: number of items hashed to bucket i. $\mathsf{S}_{i,j}$: indicator if x_j is hashed to bucket i. δ: any value ≥ 0.

Maximum Load in Randomized Hashing

$$\Pr(S_i \geq 1 + \delta) \leq \Pr\left(\left|\sum_{i=1}^{n} S_{i,j} - 1\right| \geq \delta\right) \leq 2\exp\left(-\frac{\delta^2}{2+\delta}\right).$$

Set $\delta = 20 \log m$. Gives:

$$\Pr(S_i \geq 20 \log m + 1) \leq 2\exp\left(-\frac{(20 \log m)^2}{2 + 20 \log m}\right) \leq \exp(-18 \log m) \leq \frac{2}{m^{18}}.$$

Apply Union Bound:

$$\Pr(\max_{i \in [m]} S_i \geq 20 \log m + 1) = \Pr\left(\bigcup_{i=1}^{m} (S_i \geq 20 \log m + 1)\right)$$

.

m: total number of items hashed and size of hash table. S_i: number of items hashed to bucket i. $S_{i,j}$: indicator if x_j is hashed to bucket i. δ: any value ≥ 0.

Maximum Load in Randomized Hashing

$$\Pr(S_i \geq 1+\delta) \leq \Pr\left(\left|\sum_{i=1}^{n} S_{i,j} - 1\right| \geq \delta\right) \leq 2\exp\left(-\frac{\delta^2}{2+\delta}\right).$$

Set $\delta = 20\log m$. Gives:

$$\Pr(S_i \geq 20\log m + 1) \leq 2\exp\left(-\frac{(20\log m)^2}{2+20\log m}\right) \leq \exp(-18\log m) \leq \frac{2}{m^{18}}.$$

Apply Union Bound:

$$\Pr(\max_{i\in[m]} S_i \geq 20\log m + 1) = \Pr\left(\bigcup_{i=1}^{m}(S_i \geq 20\log m + 1)\right)$$

$$\leq \sum_{i=1}^{m} \Pr(S_i \geq 20\log m + 1) \leq m \cdot \frac{2}{m^{18}} = \frac{2}{m^{17}}.$$

m: total number of items hashed and size of hash table. S_i: number of items hashed to bucket i. $S_{i,j}$: indicator if x_j is hashed to bucket i. δ: any value ≥ 0.

Maximum Load in Randomized Hashing

Upshot: If we randomly hash m items into a hash table with m entries the maximum load per bucket is $O(\log m)$ with very high probability.

Maximum Load in Randomized Hashing

Upshot: If we randomly hash m items into a hash table with m entries the maximum load per bucket is $O(\log m)$ with very high probability.

- So, even with a simple linked list to store the items in each bucket, worst case query time is $O(\log m)$.

Maximum Load in Randomized Hashing

Upshot: If we randomly hash m items into a hash table with m entries the maximum load per bucket is $O(\log m)$ with very high probability.

- So, even with a simple linked list to store the items in each bucket, worst case query time is $O(\log m)$.
- Using Chebyshev's inequality could only show the maximum load is bounded by $O(\sqrt{m})$ with good probability (good exercise).

Maximum Load in Randomized Hashing

Upshot: If we randomly hash m items into a hash table with m entries the maximum load per bucket is $O(\log m)$ with very high probability.

- So, even with a simple linked list to store the items in each bucket, worst case query time is $O(\log m)$.
- Using Chebyshev's inequality could only show the maximum load is bounded by $O(\sqrt{m})$ with good probability (good exercise).
- The Chebyshev bound holds even with a pairwise independent hash function. The stronger Chernoff-based bound can be shown to hold with a *k-wise independent hash function* for $k = O(\log m)$.

Approximately Maintaining a Set

Want to store a set S of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

Approximately Maintaining a Set

Want to store a set S of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

Goal: support $insert(x)$ to add x to the set and $query(x)$ to check if x is in the set. Both in $O(1)$ time.

Approximately Maintaining a Set

Want to store a set S of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

Goal: support $insert(x)$ to add x to the set and $query(x)$ to check if x is in the set. Both in $O(1)$ time. What data structure solves this problem?

hash tables

Approximately Maintaining a Set

Want to store a set S of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

Goal: support *insert*(x) to add x to the set and *query*(x) to check if x is in the set. Both in $O(1)$ time. What data structure solves this problem?

- Allow small probability $\delta > 0$ of false positives. I.e., for any x,

$$\Pr(query(x) = 1 \text{ and } x \notin S) \leq \delta.$$

(false positive rate

Approximately Maintaining a Set

Want to store a set S of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

Goal: support *insert*(x) to add x to the set and *query*(x) to check if x is in the set. Both in $O(1)$ time. What data structure solves this problem?

- Allow small probability $\delta > 0$ of false positives. I.e., for any x,

$$\Pr(query(x) = 1 \text{ and } x \notin S) \leq \delta.$$

Solution: Bloom filters (repeated random hashing). Will use much less space than a hash table.

Bloom Filters

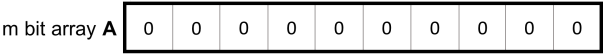
Chose k independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \to [m]$.

- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- *query*(x): return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$.

Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

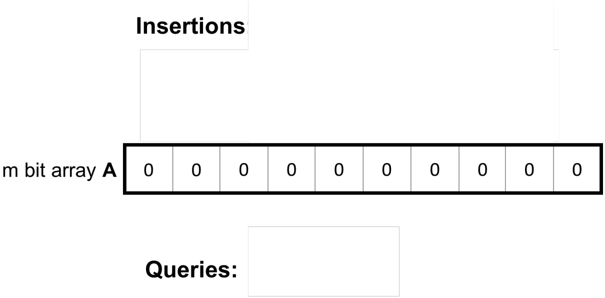
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

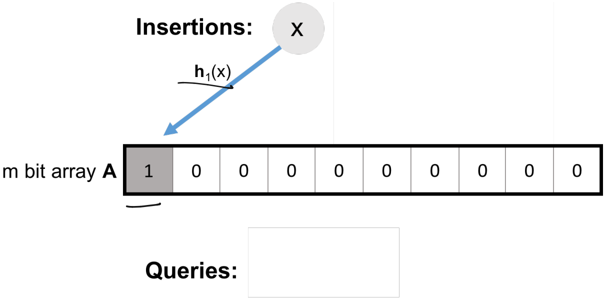
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.

Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

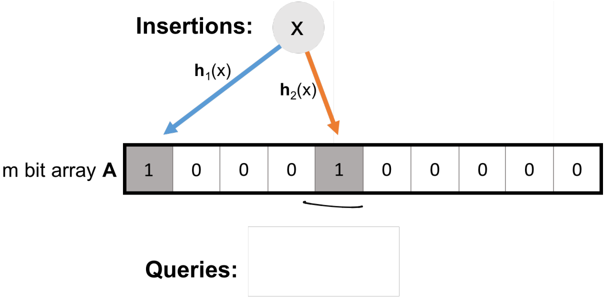
- Maintain an array A containing m bits, all initially 0.
- $insert(x)$: set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- $query(x)$: return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

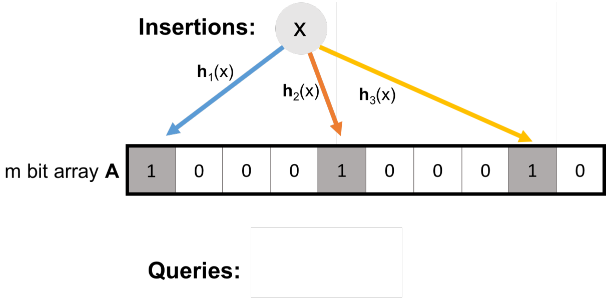
- Maintain an array A containing m bits, all initially 0.
- $insert(x)$: set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- $query(x)$: return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

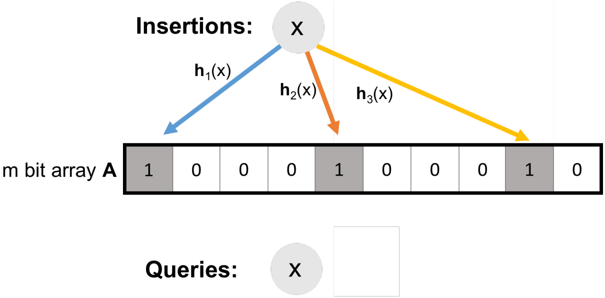
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

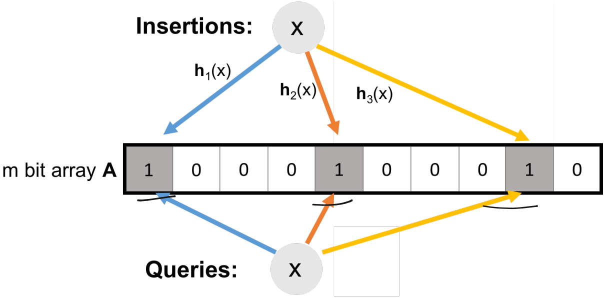
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

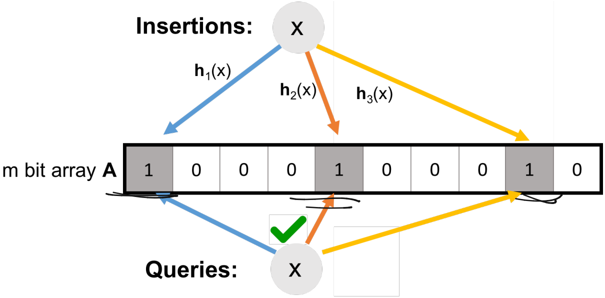
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

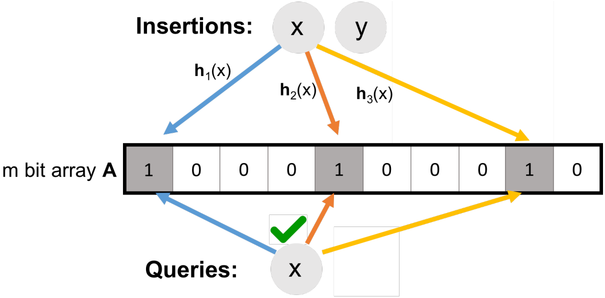
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

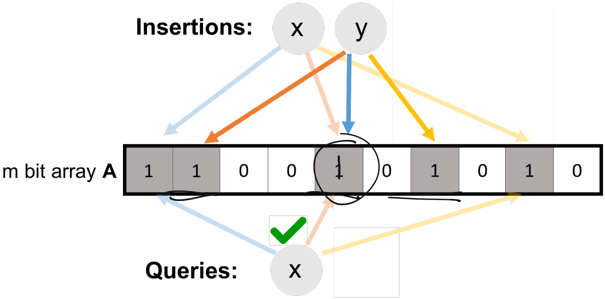
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

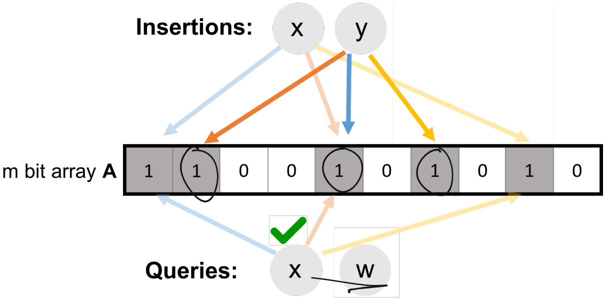
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

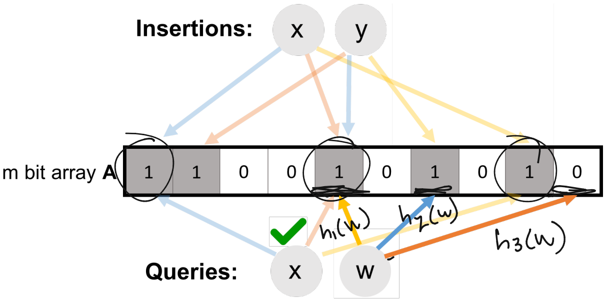
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

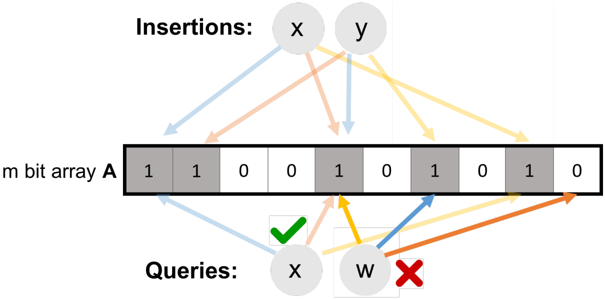
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

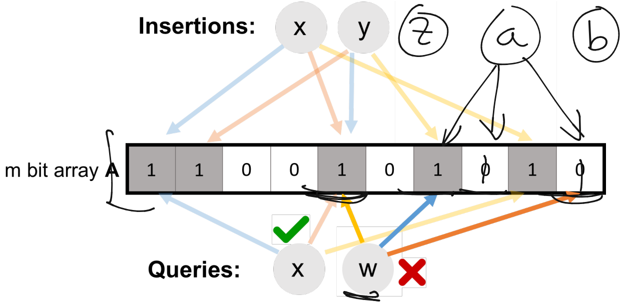
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



Bloom Filters

Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

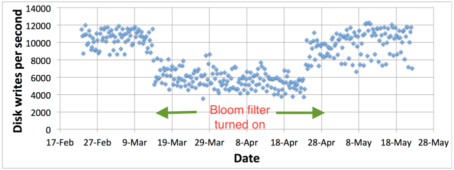
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



No false negatives. False positives more likely with more insertions.

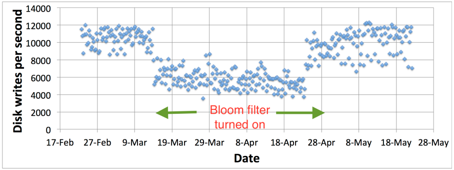
Applications: Caching

Akamai (Boston-based company serving 15 – 30% of all web traffic) applies bloom filters to prevent caching of 'one-hit-wonders' – pages only visited once fill over 75% of cache.



Applications: Caching

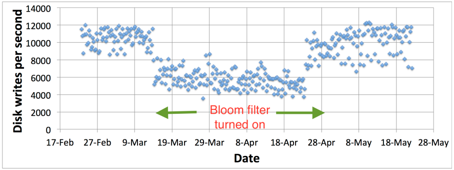
Akamai (Boston-based company serving 15 – 30% of all web traffic) applies bloom filters to prevent caching of 'one-hit-wonders' – pages only visited once fill over 75% of cache.



- When url x comes in, if $query(x) = 1$, cache the page at x. If not, run $insert(x)$ so that if it comes in again, it will be cached.

Applications: Caching

Akamai (Boston-based company serving 15 – 30% of all web traffic) applies bloom filters to prevent caching of 'one-hit-wonders' – pages only visited once fill over 75% of cache.



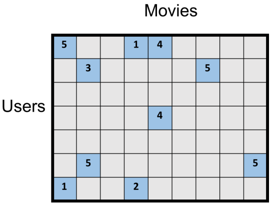
- When url x comes in, if $query(x) = 1$, cache the page at x. If not, run $insert(x)$ so that if it comes in again, it will be cached.
- **False positive:** A new url (possible one-hit-wonder) is cached. If the bloom filter has a false positive rate of $\delta = .05$, the number of cached one-hit-wonders will be reduced by at least 95%.

Applications: Databases

Distributed database systems, including Google Bigtable, Apache HBase, Apache Cassandra, and PostgreSQL use bloom filters to prevent expensive lookups of non-existent data.

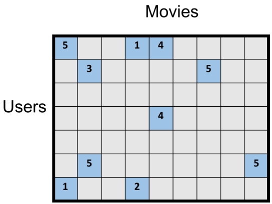
Applications: Databases

Distributed database systems, including Google Bigtable, Apache HBase, Apache Cassandra, and PostgreSQL use bloom filters to prevent expensive lookups of non-existent data.



Applications: Databases

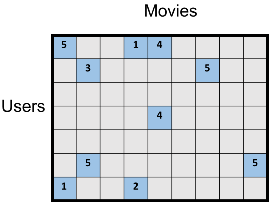
Distributed database systems, including Google Bigtable, Apache HBase, Apache Cassandra, and PostgreSQL use bloom filters to prevent expensive lookups of non-existent data.



- When a new rating is inserted for $(user_x, movie_y)$, add $(user_x, movie_y)$ to a bloom filter.
- Before reading $(user_x, movie_y)$ (possibly via an out of memory access), check the bloom filter, which is stored in memory.

Applications: Databases

Distributed database systems, including Google Bigtable, Apache HBase, Apache Cassandra, and PostgreSQL use bloom filters to prevent expensive lookups of non-existent data.



- When a new rating is inserted for $(user_x, movie_y)$, add $(user_x, movie_y)$ to a bloom filter.
- Before reading $(user_x, movie_y)$ (possibly via an out of memory access), check the bloom filter, which is stored in memory.
- **False positive:** A read is made to a possibly empty cell. A $\delta = .05$ false positive rate gives a 95% reduction in these empty reads.

More Applications

- **Database Joins:** Quickly eliminate most keys in one column that don't correspond to keys in another.
- **Recommendation systems:** Bloom filters are used to prevent showing users the same recommendations twice.
- **Spam/Fraud Detection**:
  - Bit.ly and Google Chrome use bloom filters to quickly check if a url maps to a flagged site and prevent a user from following it.
  - Can be used to detect repeat clicks on the same ad from a single IP-address, which may be the result of fraud.
- **Digital Currency:** Some Bitcoin clients use bloom filters to quickly pare down the full transaction log to transactions involving bitcoin addresses that are relevant to them (SPV: simplified payment verification).

Analysis

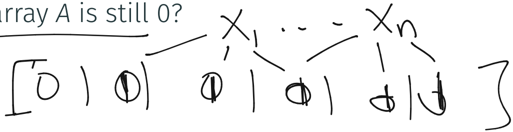
For a bloom filter with m bits and k hash functions, the insertion and query time is $O(k)$.

Analysis

For a bloom filter with m bits and k hash functions, the insertion and query time is $O(k)$. How does the false positive rate δ depend on m, k, and the number of items inserted?

For a bloom filter with m bits and k hash functions, the insertion and query time is $O(k)$. How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?



$$\left(\frac{m-1}{m}\right)^{nk}$$

Analysis

For a bloom filter with m bits and k hash functions, the insertion and query time is $O(k)$. How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0? $n \times k$ total hashes must not hit bit i.

$$\Pr(A[i] = 0) = \Pr\big(\mathbf{h}_1(x_1) \neq i \cap \ldots \cap \mathbf{h}_k(x_1) \neq i \\ \cap\, \mathbf{h}_1(x_2) \neq i \ldots \cap \mathbf{h}_k(x_2) \neq i \cap \ldots\big)$$

Analysis

For a bloom filter with m bits and k hash functions, the insertion and query time is $O(k)$. How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0? $n \times k$ total hashes must not hit bit i.

$$\begin{aligned}\Pr(A[i] = 0) = \Pr\big(&\mathbf{h}_1(x_1) \neq i \cap \ldots \cap \mathbf{h}_k(x_k) \neq i \\ &\cap \mathbf{h}_1(x_2) \neq i \ldots \cap \mathbf{h}_k(x_2) \neq i \cap \ldots\big) \\ = &\underbrace{\Pr\big(\mathbf{h}_1(x_1) \neq i\big) \times \ldots \times \Pr\big(\mathbf{h}_k(x_1) \neq i\big) \times \Pr\big(\mathbf{h}_1(x_2) \neq i\big) \ldots}_{k \cdot n \text{ events each occuring with probability } 1-1/m}\end{aligned}$$

Analysis

For a bloom filter with m bits and k hash functions, the insertion and query time is $O(k)$. How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0? $n \times k$ total hashes must not hit bit i.

$$\begin{aligned}
\Pr(A[i] = 0) &= \Pr\big(\mathbf{h}_1(x_1) \neq i \cap \ldots \cap \mathbf{h}_k(x_k) \neq i \\
&\qquad \cap\, \mathbf{h}_1(x_2) \neq i \ldots \cap \mathbf{h}_k(x_2) \neq i \cap \ldots\big) \\
&= \underbrace{\Pr\big(\mathbf{h}_1(x_1) \neq i\big) \times \ldots \times \Pr\big(\mathbf{h}_k(x_1) \neq i\big) \times \Pr\big(\mathbf{h}_1(x_2) \neq i\big) \ldots}_{k \cdot n \text{ events each occuring with probability } 1-1/m} \\
&= \left(1 - \frac{1}{m}\right)^{kn}
\end{aligned}$$

Analysis

How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn}$$

n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

Analysis

How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

Analysis

How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

Analysis

How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

$$\begin{aligned}\Pr\left(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1\right) \\ = \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1)\end{aligned}$$

n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

Analysis

How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

$$\begin{aligned}
\Pr\left(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1\right) \\
&= \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1) \\
&= \left(1 - e^{-\frac{kn}{m}}\right)^k
\end{aligned}$$

n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

Analysis

How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

$$\begin{aligned}\Pr\left(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1\right) &= \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1) \\ &= \left(1 - e^{-\frac{kn}{m}}\right)^k\end{aligned}$$

Actually Incorrect!

> n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.

Analysis

How does the false positive rate δ depend on m, k, and the number of items inserted?

Step 1: What is the probability that after inserting n elements, the i^{th} bit of the array A is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Step 2: What is the probability that querying a new item w gives a false positive?

$$\begin{aligned}\Pr\left(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1\right) &= \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1) \\ &= \left(1 - e^{-\frac{kn}{m}}\right)^k\end{aligned}$$

Actually Incorrect! Dependent events.

> n: total number items in filter, m: number of bits in filter, k: number of random hash functions, $\mathbf{h}_1, \ldots \mathbf{h}_k$: hash functions, A: bit array, δ: false positive rate.