COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Fall 2020.

Lecture 5

LOGISTICS

- Problem Set 1 is due this Friday, 9/11 at 8pm in Gradescope.
- If you can, we encourage you to make your questions public on Piazza.

Say requests were independent!

X = # failures $\Pr(X > 0) = 1 - .99^{20}$

- Problem Set 1 is due this Friday, 9/11 at 8pm in Gradescope.
- If you can, we encourage you to make your questions public on Piazza.

$1 - .99^{20}$ – we don't have independence

Quiz 2:

- Class Pace: 48% just right, 42% a bit too fast, 5% a bit too slow, 5% way too fast.

$A_1, A_2 \ldots A_{20}$

- I receive 20 download requests per day and serve each in within 15 seconds with probability 99%. Upper bound the probability I *fail to serve at least one request.*

$P(A_i) = .01$ $\Pr(A_1 \cup A_2 \cup \ldots A_{20}) \leq \sum P(A_i) = 20 \cdot .01 = .02$

Last Class: Concentration bounds beyond Markov's inequality

- Chebyshev's inequality and the law of large numbers.
- Exponential concentration bounds from higher moments.
- Bernstein's Inequality

Last Class: Concentration bounds beyond Markov's inequality

- Chebyshev's inequality and the law of large numbers.
- Exponential concentration bounds from higher moments.
- Bernstein's Inequality

This Time:

- Finish up exponential concentration bounds and the central limit theorem.

- Start on algorithms: Bloom Filters

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:

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

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:

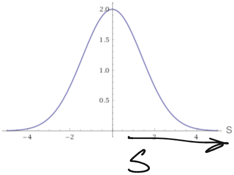
$$\Pr\left(\left|\sum_{i=1}^{n} X_i - \mu\right| \geq s\sigma\right) \leq 2\exp\left(-\frac{s^2}{4}\right).$$

Can plot this bound for different s:

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:

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

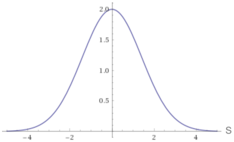
Can plot this bound for different s:



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:

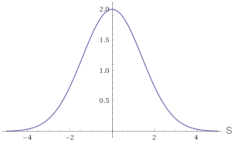


Looks a lot like a Gaussian (normal) distribution.

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:



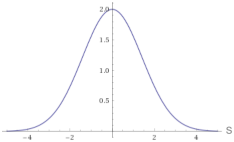
Looks a lot like a Gaussian (normal) distribution.

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

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.

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

GAUSSIAN TAILS

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

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

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

$$\Pr\left(|X| \geq s \cdot \sigma\right) \leq O(1) \cdot e^{-\frac{s^2}{2}}.$$

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

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

$$\Pr\left(|X| \geq s \cdot \sigma\right) \leq O(1) \cdot e^{-\frac{s^2}{2}}.$$

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

Essentially the same bound that Bernstein's inequality gives!

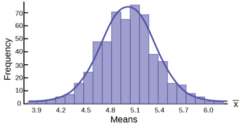
$$\mathcal{N}(0, \sigma^2) \text{ has density } p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{x^2}{2\sigma^2}}.$$

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

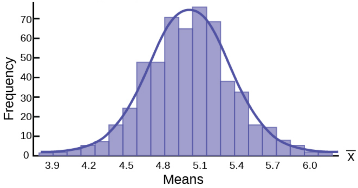
$$\Pr\left(|\mathbf{X}| \geq s \cdot \sigma\right) \leq O(1) \cdot e^{-\frac{s^2}{2}}.$$

Essentially the same bound that Bernstein's inequality gives!

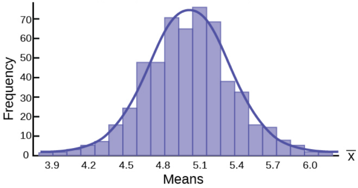
Central Limit Theorem Interpretation: Bernstein's inequality gives a quantitative version of the CLT. The distribution of the sum of *bounded* independent random variables can be upper bounded with a Gaussian (normal) distribution.



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.

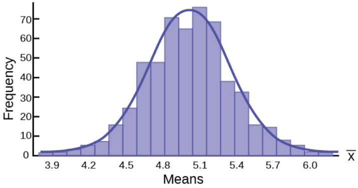


Stronger Central Limit Theorem: The distribution of the sum of *n bounded* independent random variables converges to a Gaussian (normal) distribution as *n* goes to infinity.



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

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 $\delta \geq 0$

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

THE CHERNOFF BOUND

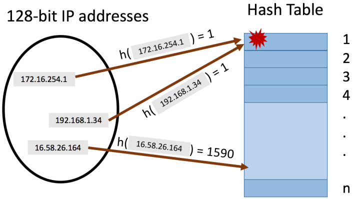
A useful variation of the Bernstein inequality for binary (indicator) random variables is:

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 $\delta \geq 0$

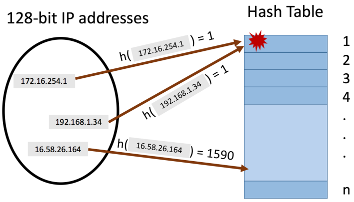
$$\Pr\left(\left|\sum_{i=1}^{n} X_i - \mu\right| \geq \delta\mu\right) \leq 2\exp\left(-\frac{\delta^2\mu}{2+\delta}\right).$$

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

$n = m^2$
collision free
2 level hashing
$n \sim m$

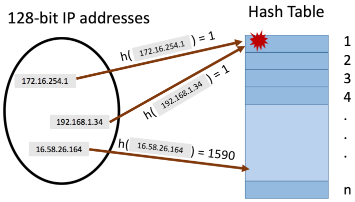


We hash m values $x_1, \ldots, x_m$ using a random hash function into a table with $n = m$ entries.



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 $i \in [n]$, $\Pr(\mathbf{h}(x) = i) = \frac{1}{m}$ and hash values are chosen independently.



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 $i \in [n]$, $\Pr(\mathbf{h}(x) = i) = \frac{1}{m}$ and hash values are chosen independently.

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

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

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

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

$$\mathbb{E}[S_i] = \sum_{j=1}^{m} \mathbb{E}[S_{i,j}] = m \cdot \frac{1}{m} = 1$$

$\frac{1}{m}$

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

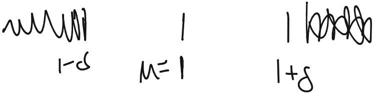
Let S_i be the number of items hashed into position i and $\mathsf{S}_{i,j}$ be 1 if x_j is hashed into bucket i ($\mathsf{h}(x_j) = i$) and 0 otherwise.

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

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

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

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



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

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

m: total number of items hashed and size of hash table. $x_1, \ldots, x_m$: the items.
$\mathbf{h}$: 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

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

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

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

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

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

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

$\approx 20\log m$

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.

$\delta = 10 \quad Pr(S_i \geq 11) \leq 2e^{-\frac{10^2}{12}} \approx 2e^{-8}$

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

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

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

$(e^{-\log m})^{18} = \frac{1}{m^{18}}$

Apply Union Bound:

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

$$\leq m \cdot \frac{2}{m^{18}} = \frac{2}{m^{17}}$$

m: total number of items hashed and size of hash table. S_i: number of items hashed to bucket i. $\mathsf{S}_{i,j}$: 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:

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

Apply Union Bound:

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

$$\leq \sum_{i=1}^{m} \Pr(\mathsf{S}_i \geq 20\log m + 1) \qquad .$$

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

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

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

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

Apply Union Bound:

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

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

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

20 log m + 1

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.

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 $O(\log m)$.

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

$X_1, X_2 \dots X_n$

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 $k = O(\log m)$.

Questions on Exponential Concentration Bounds?

This concludes the probability foundations part of the course – on to algorithms.

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

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.

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 table

n items

O(n)

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,

$\delta = .01$

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

cuckoo hash

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.

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

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.

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

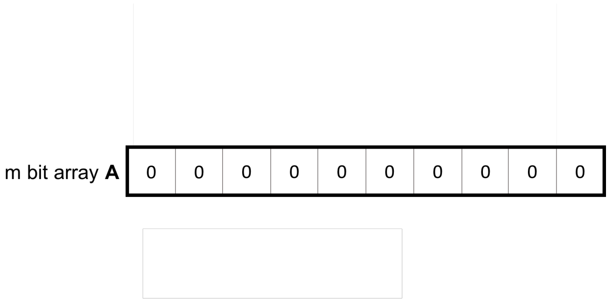
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathrm{h}_1(x)] = \ldots = A[\mathrm{h}_k(x)] := 1$.

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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.

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

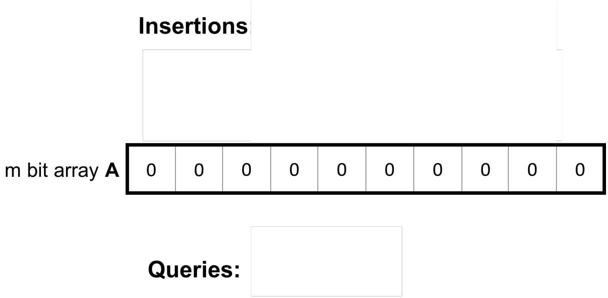
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



BLOOM FILTERS

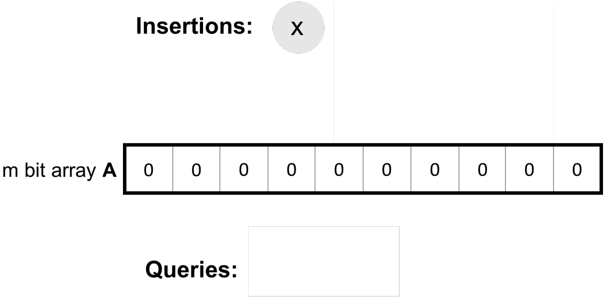
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



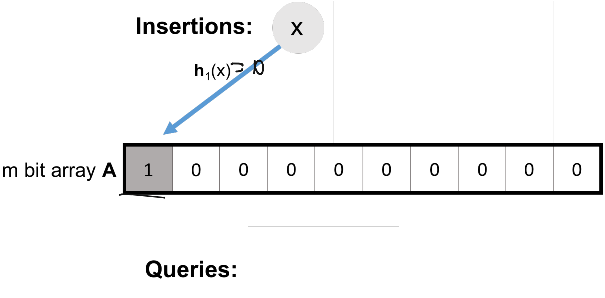
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



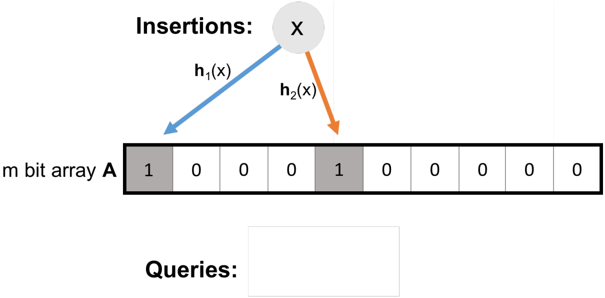
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



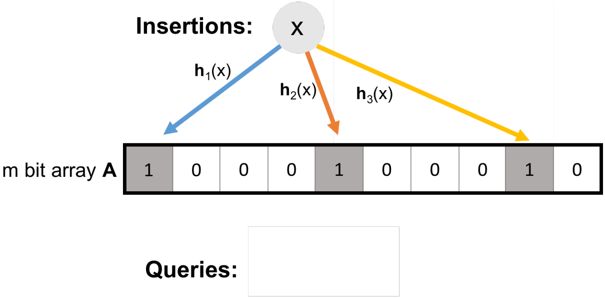
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



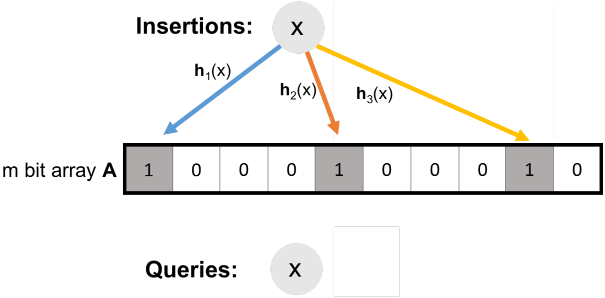
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



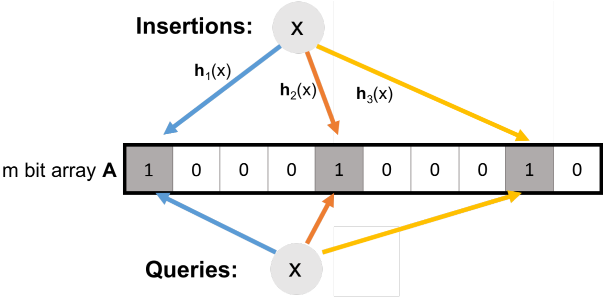
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



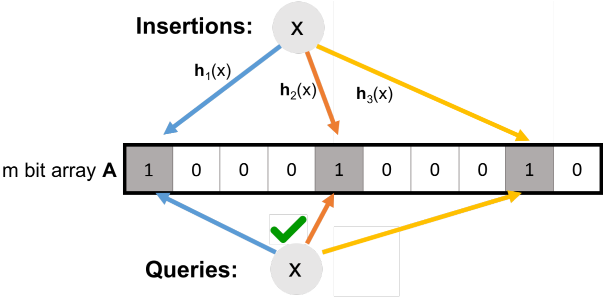
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



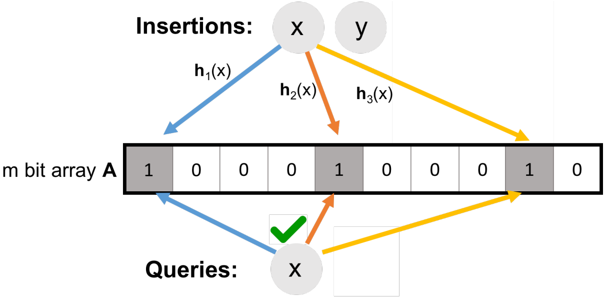
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



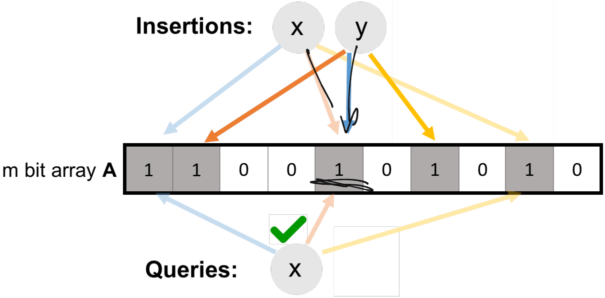
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



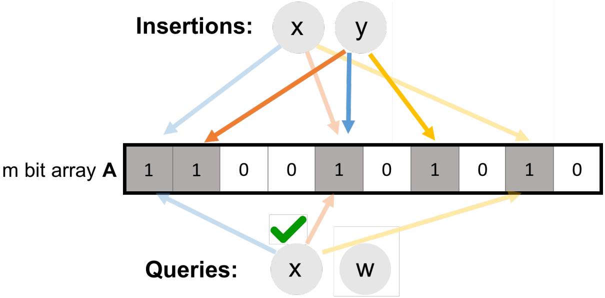
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



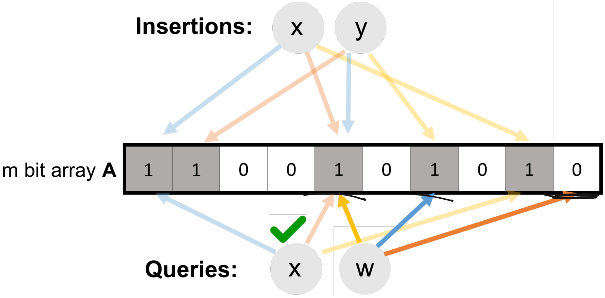
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



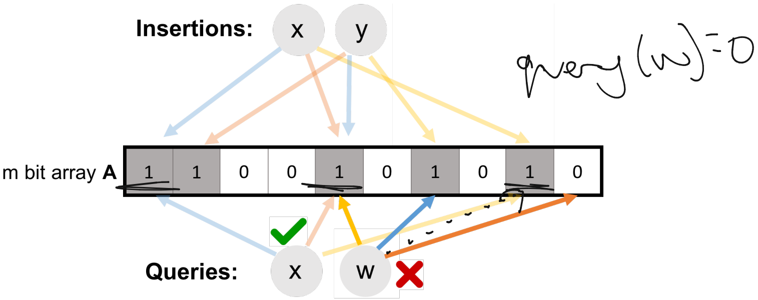
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



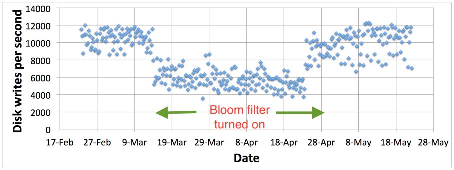
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- 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 $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



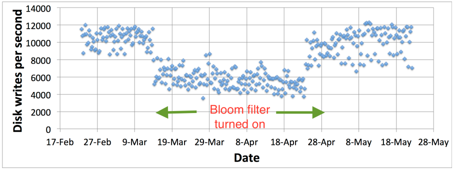
No false negatives. False positives more likely with more insertions.

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

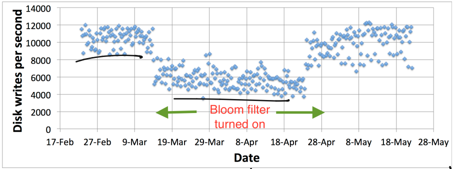
Akamai (Boston-based company serving 15 – 30% of all web traffic) applies bloom filters to prevent caching of ‘one-hit-wonders’ – pages only visited once fill over 75% of cache.



- 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.

count-min sketch

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.



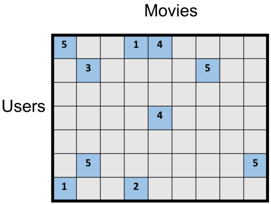
x has never been seen by query(x) = 1 w.p. δ

- 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

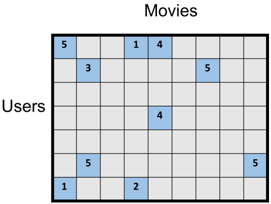
Distributed database systems, including Google Bigtable, Apache HBase, Apache Cassandra, and PostgreSQL use bloom filters to prevent expensive lookups of non-existent data.

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

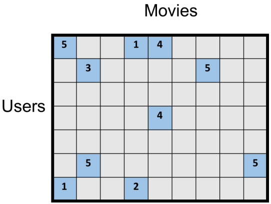
Distributed database systems, including Google Bigtable, Apache HBase, Apache Cassandra, and PostgreSQL use bloom filters to prevent expensive lookups of non-existent data.



- 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 requiring 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 requiring 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.

Bloom filters are used by Oracle and other database companies to speed up database *joins*.

Bloom filters are used by Oracle and other database companies to speed up database *joins*.

INNER JOIN

Customers

CustomerId	Name
1	Robert
2	Peter
3	Smith

Orders

OrderId	CustomerId	OrderDate
100	1	2016-10-19 15:21:27
200	4	2016-10-20 15:21:27
300	2	2016-10-21 15:21:27

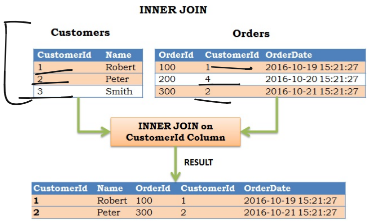
INNER JOIN on CustomerId Column

RESULT

CustomerId	Name	OrderId	CustomerId	OrderDate
1	Robert	100	1	2016-10-19 15:21:27
2	Peter	300	2	2016-10-21 15:21:27

- Matches up a key in column A of one table to a key in column B of another, and merges corresponding information.

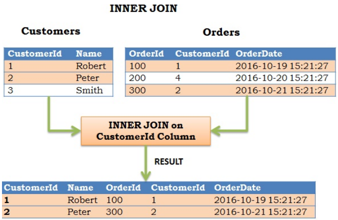
Bloom filters are used by Oracle and other database companies to speed up database *joins*.



- Matches up a key in column **A** of one table to a key in column **B** of another, and merges corresponding information.
- A bloom filter can be used to quickly eliminate entries that appear in **A** but not in **B**.

APPLICATIONS: DATABASES

Bloom filters are used by Oracle and other database companies to speed up database *joins*.



- Matches up a key in column **A** of one table to a key in column **B** of another, and merges corresponding information.
- A bloom filter can be used to quickly eliminate entries that appear in **A** but not in **B**.
- A false positive rate of δ means that a $1 - \delta$ fraction of these entries can be eliminated in the initial bloom filter check.

- Recommendation systems (Netflix, Youtube, Tinder, etc.) use bloom filters to prevent showing users the same recommendations twice.

- Recommendation systems (Netflix, Youtube, Tinder, etc.) use bloom filters 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.

MORE APPLICATIONS

- **Recommendation systems** (Netflix, Youtube, Tinder, etc.) use bloom filters 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).

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

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?

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?

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.

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

n items, k hashes per item

$h_1(x_1) \neq i$ $\quad \left(1 - \frac{1}{m}\right)^{kn}$

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.

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

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.

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

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.

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?

$$\lim_{m \to \infty} \left(1 - \frac{1}{m}\right)^{m} = e$$

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

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.

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.

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.

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.

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.

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.

Step 1: To avoid dependence issues, condition on the event that the A has t zeros in it after n insertions, for some $t \leq m$. For a non-inserted element w, after conditioning on this event we correctly have:

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

I.e., the events $A[\mathbf{h}_1(w)] = 1$,..., $A[\mathbf{h}_k(w)] = 1$ are independent conditioned on the number of bits set in A.

Step 1: To avoid dependence issues, condition on the event that the A has t zeros in it after n insertions, for some $t \leq m$. For a non-inserted element w, after conditioning on this event we correctly have:

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

I.e., the events $A[\mathbf{h}_1(w)] = 1$,..., $A[\mathbf{h}_k(w)] = 1$ are independent conditioned on the number of bits set in A. Why?

CORRECT ANALYSIS SKETCH

Step 1: To avoid dependence issues, condition on the event that the A has t zeros in it after n insertions, for some $t \leq m$. For a non-inserted element w, after conditioning on this event we correctly have:

$$\Pr(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1) \\ = \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1).$$

I.e., the events $A[\mathbf{h}_1(w)] = 1$,..., $A[\mathbf{h}_k(w)] = 1$ are independent conditioned on the number of bits set in A. Why?

- Conditioned on this event, for any j, since $\mathbf{h}_j$ is a fully random hash function, $\Pr(A[\mathbf{h}_j(w)] = 1) = \frac{t}{m}$.

CORRECT ANALYSIS SKETCH

Step 1: To avoid dependence issues, condition on the event that the A has t zeros in it after n insertions, for some $t \leq m$. For a non-inserted element w, after conditioning on this event we correctly have:

$$\Pr(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1)$$
$$= \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1).$$

I.e., the events $A[\mathbf{h}_1(w)] = 1$,..., $A[\mathbf{h}_k(w)] = 1$ are independent conditioned on the number of bits set in A. Why?

- Conditioned on this event, for any j, since $\mathbf{h}_j$ is a fully random hash function, $\Pr(A[\mathbf{h}_j(w)] = 1) = \frac{t}{m}$.
- Thus conditioned on this event, the false positive rate is $\left(1 - \frac{t}{m}\right)^k$.

Step 1: To avoid dependence issues, condition on the event that the A has t zeros in it after n insertions, for some $t \leq m$. For a non-inserted element w, after conditioning on this event we correctly have:

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

I.e., the events $A[\mathbf{h}_1(w)] = 1$,..., $A[\mathbf{h}_k(w)] = 1$ are independent conditioned on the number of bits set in A. Why?

- Conditioned on this event, for any j, since $\mathbf{h}_j$ is a fully random hash function, $\Pr(A[\mathbf{h}_j(w)] = 1) = \frac{t}{m}$.
- Thus conditioned on this event, the false positive rate is $\left(1 - \frac{t}{m}\right)^k$.
- It remains to show that $\frac{t}{m} \approx e^{-\frac{kn}{m}}$ with high probability. We already have that $\mathbb{E}[\frac{t}{m}] = \frac{1}{m}\sum_{i=1}^{m} \Pr(A[i] = 0) \approx e^{-\frac{kn}{m}}$.

Need to show that the number of zeros t in A after n insertions is bounded by $O\left(e^{-\frac{kn}{m}}\right)$ with high probability.

Can apply Theorem 2 of: `http://cglab.ca/~morin/publications/ds/bloom-submitted.pdf`

Questions on Bloom Filters?