COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Andrew McGregor

Lecture 5

A variation of the Bernstein inequality for binary random variables is:

Chernoff Bound (simplified version): Consider independent random variables $\mathbf{X}_1, \ldots, \mathbf{X}_n$ taking values in $\{0, 1\}$. Let $\mu = \mathbb{E}[\sum_{i=1}^n \mathbf{X}_i]$. For any $\delta \geq 0$

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

THE CHERNOFF BOUND

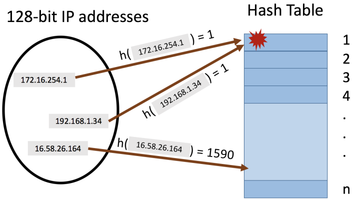
A variation of the Bernstein inequality for binary random variables is:

Chernoff Bound (simplified version): Consider independent random variables $\mathbf{X}_1, \ldots, \mathbf{X}_n$ taking values in $\{0, 1\}$. Let $\mu = \mathbb{E}[\sum_{i=1}^n \mathbf{X}_i]$. For any $\delta \geq 0$

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

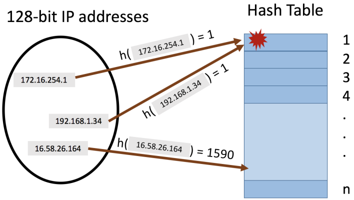
As δ gets larger and larger, the bound falls of exponentially fast.

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.

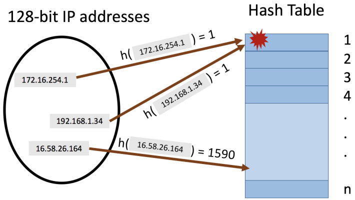
RETURN TO RANDOM HASHING



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

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

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

MAXIMUM LOAD IN RANDOMIZED HASHING

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 $\mathbf{S}_i$ 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.

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

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

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.

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

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 $\mathbf{S}_i$ 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 $\delta \geq 0$,

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

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

MAXIMUM LOAD IN RANDOMIZED HASHING

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

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

MAXIMUM LOAD IN RANDOMIZED HASHING

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

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

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.

MAXIMUM LOAD IN RANDOMIZED HASHING

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

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

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

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.

MAXIMUM LOAD IN RANDOMIZED HASHING

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

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

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

Apply Union Bound:

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

.

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.

MAXIMUM LOAD IN RANDOMIZED HASHING

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

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

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

Apply Union Bound:

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

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

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.

MAXIMUM LOAD IN RANDOMIZED HASHING

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

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

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

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.

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 $O(\log m)$.

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).

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...

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.

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.

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

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

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.

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

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

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 $U \to [m]$.

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.

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

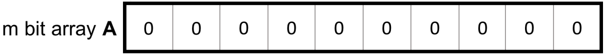
BLOOM FILTERS

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

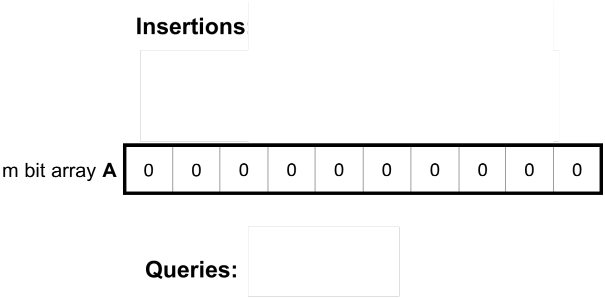
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

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



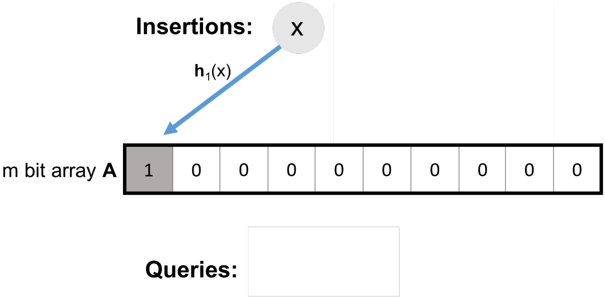
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

- 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 $U \to [m]$.

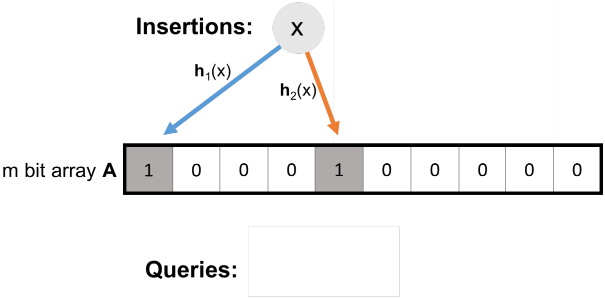
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



BLOOM FILTERS

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

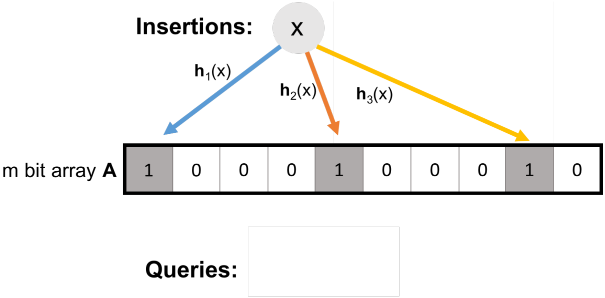
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



BLOOM FILTERS

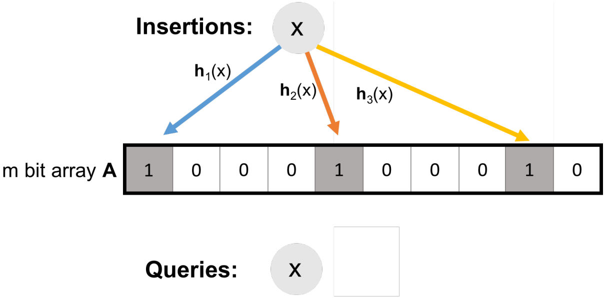
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

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



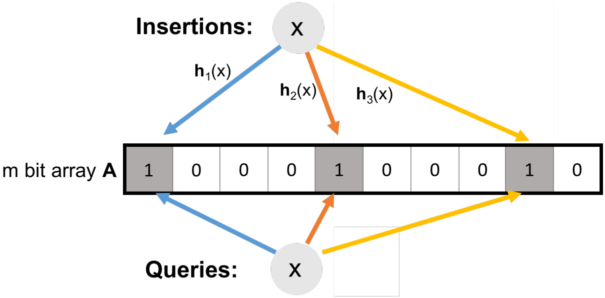
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

- 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 $U \rightarrow [m]$.

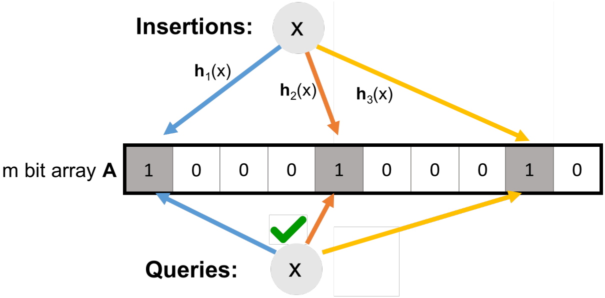
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



BLOOM FILTERS

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

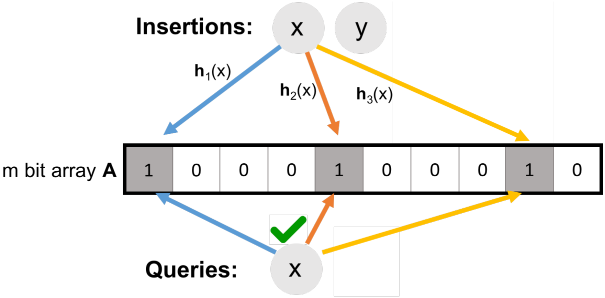
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



BLOOM FILTERS

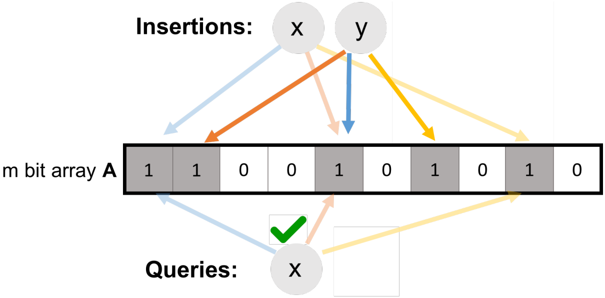
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

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



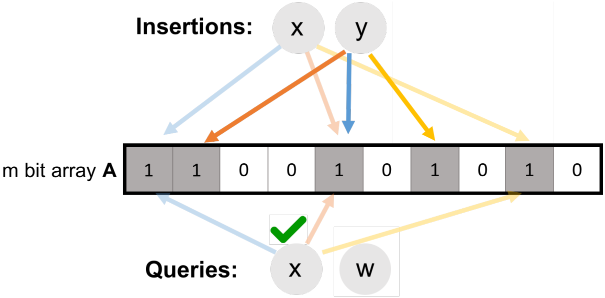
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

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



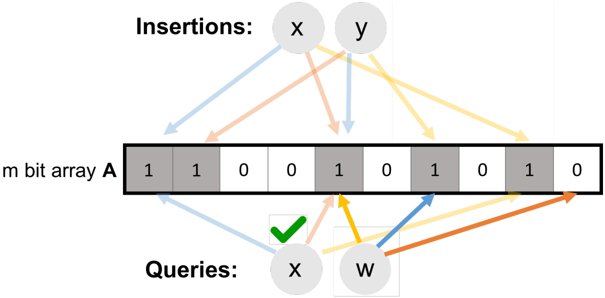
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \to [m]$.

- 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 $U \to [m]$.

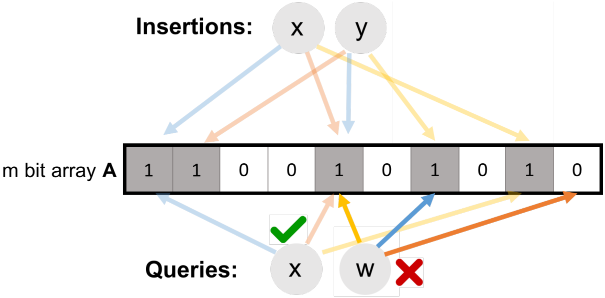
- Maintain an array A containing m bits, all initially 0.
- *insert*(x): set all bits $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] := 1$.
- *query*(x): return 1 only if $A[\mathbf{h}_1(x)] = \ldots = A[\mathbf{h}_k(x)] = 1$.



BLOOM FILTERS

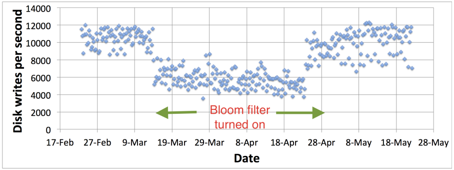
Chose k independent random hash functions $\mathbf{h}_1, \ldots, \mathbf{h}_k$ mapping the universe of elements $U \rightarrow [m]$.

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



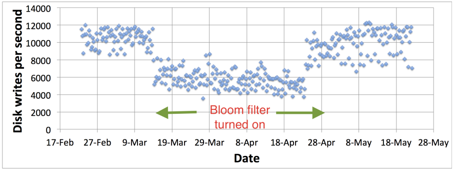
No false negatives. False positives more likely with more insertions.

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: DETERMINING WHEN TO CACHE

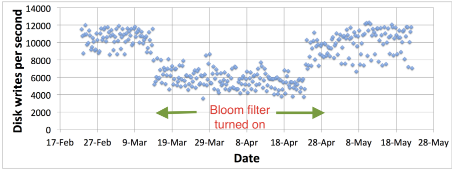
Akamai (Boston-based company serving 15 – 30% of all web traffic) applies bloom filters to prevent caching of 'one-hit-wonders' – pages only visited once fill over 75% of cache.



- A Bloom Filter can be used to approximately track the url's you've seen before without have to store them all!

APPLICATIONS: DETERMINING WHEN TO CACHE

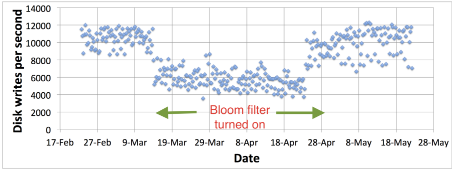
Akamai (Boston-based company serving 15 – 30% of all web traffic) applies bloom filters to prevent caching of 'one-hit-wonders' – pages only visited once fill over 75% of cache.



- A Bloom Filter can be used to approximately track the url's you've seen before without have to store them all! When url x comes in, if $query(x) = 1$, cache the page if it isn't already cached. If not, run $insert(x)$ so that if it comes in again, it will be cached.

APPLICATIONS: DETERMINING WHEN TO CACHE

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.



- A Bloom Filter can be used to approximately track the url's you've seen before without have to store them all! When url x comes in, if $query(x) = 1$, cache the page if it isn't already cached. 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%.

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.

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

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.

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

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.

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

ANALYSIS

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

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?

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.

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

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

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

Let T be the number of zeros in the array after n inserts. Then,

$$E[T] = m\left(1 - \frac{1}{m}\right)^{kn} \approx me^{-\frac{kn}{m}}$$

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.

CORRECT ANALYSIS SKETCH

If T is the number of 0 entries, for a non-inserted element w:

$$
\begin{aligned}
&\Pr(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1) \\
&= \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1) \\
&= (1 - T/m) \times \ldots \times (1 - T/m) \\
&= (1 - T/m)^k
\end{aligned}
$$

CORRECT ANALYSIS SKETCH

If T is the number of 0 entries, for a non-inserted element w:

$$\begin{aligned}
&\Pr(A[\mathbf{h}_1(w)] = \ldots = A[\mathbf{h}_k(w)] = 1) \\
&= \Pr(A[\mathbf{h}_1(w)] = 1) \times \ldots \times \Pr(A[\mathbf{h}_k(w)] = 1) \\
&= (1 - T/m) \times \ldots \times (1 - T/m) \\
&= (1 - T/m)^k
\end{aligned}$$

- How small is T/m? Note that $\frac{T}{m} \geq \frac{m-nk}{m} \approx e^{-\frac{kn}{m}}$ when $kn \ll m$. More generally, it can be shown that $T/m = \Omega\left(e^{-\frac{kn}{m}}\right)$ via Theorem 2 of:

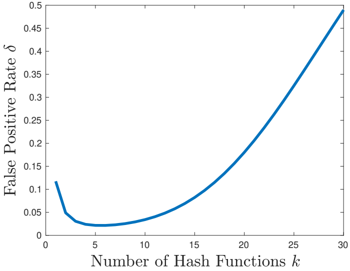
  `cglab.ca/~morin/publications/ds/bloom-submitted.pdf`

FALSE POSITIVE RATE

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$.

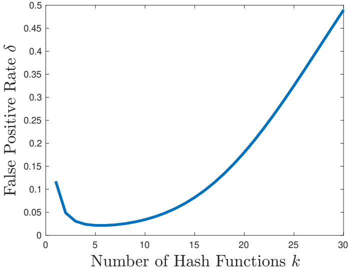
FALSE POSITIVE RATE

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$.



FALSE POSITIVE RATE

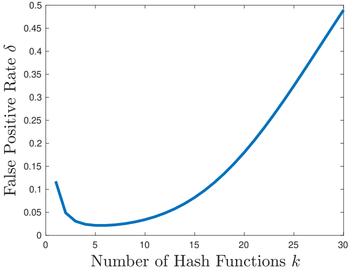
False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$.



- Can differentiate to show optimal number of hashes is $k = \ln 2 \cdot \frac{m}{n}$ (rounded to the nearest integer). This gives $\delta \approx 1/2^{(m/n)\ln 2}$.

FALSE POSITIVE RATE

False Positive Rate: with m bits of storage, k hash functions, and n items inserted $\delta \approx \left(1 - e^{\frac{-kn}{m}}\right)^k$.



- Can differentiate to show optimal number of hashes is $k = \ln 2 \cdot \frac{m}{n}$ (rounded to the nearest integer). This gives $\delta \approx 1/2^{(m/n)\ln 2}$.
- Balances between filling up the array with too many hashes and having enough hashes so that even when the array is pretty full, a new item is unlikely to have all its bits set (yield a false positive)