3.3 Variance and Standard Deviation Recap

Anna Karlin
Most Slides by Alex Tsun
Agenda

- **Variance**
- **Independence of random variables**
- **Properties of variance**
**Variance and Standard Deviation (SD)**

**Variance:** The variance of a random variable $X$ is

$$Var(X) = E[(X - E[X])^2] = E[X^2] - (E[X])^2$$

The variance is always nonnegative since we take an expectation of a nonnegative random variable $(X - E[X])^2$. We can also show that for any scalars $a, b \in \mathbb{R}$,

$$Var(aX + b) = a^2 Var(X)$$

**Standard Deviation (SD):** The standard deviation of a random variable $X$ is

$$\sigma_X = \sqrt{Var(X)}$$

We want this because the units of variance are squared in terms of the original variable $X$, and this “undo’s” our squaring, returning the units to the same as $X$. 
Random variables and independence

**Random variable X and event E are independent** if the event E is independent of the event \(\{X=x\}\) (for any fixed \(x\)), i.e.

\[ \forall x \ P(X = x \text{ and } E) = P(X=x) \cdot P(E) \]

**Two random variables X and Y are independent** if the events \(\{X=x\}\) and \(\{Y=y\}\) are independent for any fixed \(x, y\), i.e.

\[ \forall x, y \ P(X = x \text{ and } Y=y) = P(X=x) \cdot P(Y=y) \]

Intuition as before: knowing \(X\) doesn’t help you guess \(Y\) or \(E\) and vice versa.
INDEPENDENT VS DEPENDENT R.V.S

- Dependent r.v.s can reinforce/cancel/correlate in arbitrary ways.
- Independent r.v.s are, well, independent.

Example:

\[ Z = X_1 + X_2 + \ldots + X_n \]

\( X_i \) is indicator r.v. with probability 1/2 of being 1.

versus

\[ W = n \, X_1 \]
**Important facts about independent random variables**

**Theorem:** If $X$ & $Y$ are independent, then $E[X\cdot Y] = E[X] \cdot E[Y]$

**Theorem:** If $X$ and $Y$ are independent, then

$\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y]$

**Corollary:** If $X_1 + X_2 + \ldots + X_n$ are mutually independent then

$\text{Var}[X_1 + X_2 + \ldots + X_n] = \text{Var}[X_1] + \text{Var}[X_2] + \ldots + \text{Var}[X_n]$
**E[XY] for Independent Random Variables**

- **Theorem:** If $X$ & $Y$ are independent, then $E[X \cdot Y] = E[X] \cdot E[Y]$
- **Proof:**

Let $x_i, y_i, i = 1, 2, \ldots$ be the possible values of $X, Y$.

\[
E[X \cdot Y] = \sum_i \sum_j x_i \cdot y_j \cdot P(X = x_i \land Y = y_j)
\]

\[
= \sum_i \sum_j x_i \cdot y_j \cdot P(X = x_i) \cdot P(Y = y_j) \quad \text{(independence)}
\]

\[
= \sum_i x_i \cdot P(X = x_i) \cdot \left( \sum_j y_j \cdot P(Y = y_j) \right)
\]

\[
= E[X] \cdot E[Y]
\]

**Note:** NOT true in general; see earlier example $E[X^2] \neq E[X]^2$
Variance of a sum of independent r.v.s

Theorem: If $X$ and $Y$ are independent, then

$$\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y]$$

Proof:

$$\text{Var}[X + Y]$$

$$= E[(X + Y)^2] - (E[X + Y])^2$$

$$= E[X^2 + 2XY + Y^2] - (E[X] + E[Y])^2$$


$$= E[X^2] - (E[X])^2 + E[Y^2] - (E[Y])^2 + 2(E[XY] - E[X]E[Y])$$

$$= \text{Var}[X] + \text{Var}[Y] + 2(E[X]E[Y] - E[X]E[Y])$$

$$= \text{Var}[X] + \text{Var}[Y]$$
Probability

Alex Tsun

Joshua Fan
BLOOM FILTERS

Anna Karlin
Most slides by Shreya Jayaraman, Luxi Wang, Alex Tsun
Hashing
**Basic Problem**

**Problem:** Store a subset $S$ of a large set $U$.

**Example.** $U =$ set of 128 bit strings  
$S =$ subset of strings of interest

Two goals:  
1. **Constant-time** answering of queries “Is $x \in S$?”  
2. **Minimize storage** requirements.
Naïve Solution – Constant Time

Idea: Represent $S$ as an array $A$ with $2^{128}$ entries.

$S = \{0, 2, \ldots, K\}$

Storage: Require storing $2^{64}$ bits, even for small $S$.

Membership test: To check $x \in S$ just check whether $A[x] = 1$.

→ constant time! 👍 😊

A[x] = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases}
Naïve Solution – Small Storage

Idea: Represent $S$ as a list with $|S|$ entries.

$S = \{0, 2, \ldots, K\}$

Storage: Grows with $|S|$ only

Membership test: Check $x \in S$ requires time linear in $|S|$
(Can be made logarithmic by using a tree)
Hash Table

Idea: Map elements in $S$ into an array $A$ using a hash function

Membership test: To check $x \in S$ just check whether $A[h(x)] = x$

Storage: $n$ elements
Hash Table

Idea: Map elements in \( S \) into an array \( A \) using a hash function \( h \).

Membership test: To check \( x \in S \) just check whether \( A[h(x)] = x \).

Storage: \( n \) elements

Challenge 1: Ensure \( h(x) \neq h(y) \) for most \( x, y \in S \).

Challenge 2: Ensure \( n = O(|S|) \).
Collisions occur when two elements of set map to the same location in the hash table.

Common solution: chaining – at each location (bucket) in the table, keep linked list of all elements that hash there.

Want: hash function that distributes the elements of $S$ well across hash table locations. Ideally uniform distribution!
Hash Tables

- They store the data itself
- With a good hash function, the data is well distributed in the table and lookup times are small.
- However, they need at least as much space as all the data being stored
- E.g. storing strings, or IP addresses or long DNA sequences.
Bloom Filters: Motivation

- Large universe of possible data items.
- Data items are large (say 128 bits or more)
- Hash table is stored on disk or across network, so any lookup is expensive.
- Many (if not nearly all) of the lookups return “Not found”.

Altogether, this is bad. You’re wasting a lot of time and space doing lookups for items that aren’t even present.
Bloom Filters: Motivation

- Large universe of possible data items.
- Hash table is stored on disk or in network, so any lookup is expensive.
- Many (if not most) of the lookups return “Not found”.

Altogether, this is bad. You’re wasting a lot of time and space doing lookups for items that aren’t even present.

Examples:
- Google Chrome: wants to warn you if you’re trying to access a malicious URL. Keep hash table of malicious URLs.
- Network routers: want to track source IP addresses of certain packets, e.g., blocked IP addresses.
Bloom Filters: Motivation

- Probabilistic data structure.
- Close cousins of hash tables.
- Ridiculously space efficient
- To get that, make occasional errors, specifically false positives.

Typical implementation: only 8 bits per element!
Bloom Filters
Bloom Filters

- Stores information about a set of elements.
- Supports two operations:
  1. add(x) - adds x to bloom filter
  2. contains(x) - returns true if x in bloom filter, otherwise returns false
     a. If return false, **definitely** not in bloom filter.
     b. If return true, **possibly** in the structure (some false positives).
Bloom Filters

- Why accept false positives?
  - **Speed** – both operations very very fast.
  - **Space** – requires a miniscule amount of space relative to storing all the actual items that have been added.

- Often just 8 bits per inserted item!
**Bloom Filters: Initialization**

```plaintext
function \text{INITIALIZE}(k, m) 
  for i = 1, \ldots, k: do 
  \( t_i = \text{new bit vector of m 0's} \)
```

- Number of hash functions
- Size of array associated to each hash function.
- For each hash function, initialize an empty bit vector of size m
Bloom Filters: Example

Bloom filter \( t \) with \( m = 5 \) that uses \( k = 3 \) hash functions

**Function** \( \text{initialize}(k, m) \)

\[
\text{for } i = 1, \ldots, k: \text{ do} \\
\quad t_i = \text{new bit vector of } m \text{ 0's}
\]

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Bloom Filters: Add

function ADD(x)
for $i = 1, \ldots, k$: do
$$t_i[h_i(x)] = 1$$

$h_i(x) \rightarrow$ result of hash function $h_i$ on $x$

for each hash function $h_i$
Bloom Filters: Add

```plaintext
function ADD(x)
  for i = 1, ..., k: do
    ti[h_i(x)] = 1
```

for each hash function $h_i$

Index into $i$th bit-vector, at index produced by hash function and set to 1
Bloom Filters: Example

bloom filter \( t \) with \( m = 5 \) that uses \( k = 3 \) hash functions

add("thisisavirus.com")

function `ADD(x)`

\[
\text{for } i = 1, \ldots, k: \text{ do} \\
\quad t_i[h_i(x)] = 1
\]

<table>
<thead>
<tr>
<th>Index ( t )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Bloom Filters: Example**

Bloom filter \( t \) of length \( m = 5 \) that uses \( k = 3 \) hash functions

**function** \( \text{ADD}(x) \)

\[
\text{for } i = 1, \ldots, k: \text{ do} \\
\quad t_i[h_i(x)] = 1
\]

\( \text{add(“thisisavirus.com”) → 2} \)

- \( h_1(“thisisavirus.com”) \rightarrow 2 \)

<table>
<thead>
<tr>
<th>Index →</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Bloom Filters: Example**

Bloom filter $t$ of length $m = 5$ that uses $k = 3$ hash functions

**Function** $\text{ADD}(X)$

```
for $i = 1, \ldots, k$: do
    $t_i[h_i(x)] = 1$
```

```plaintext
add("thisisavirus.com")
  $h_1("thisisavirus.com") \rightarrow 2$
  $h_2("thisisavirus.com") \rightarrow 1$
```

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Bloom Filters: Example

bloom filter \( t \) of length \( m = 5 \) that uses \( k = 3 \) hash functions

\[
\text{function } ADD(X) \\
\text{for } i = 1, \ldots, k: \ 	ext{do} \\
\quad t_i[h_i(x)] = 1
\]

add(“thisisavirus.com”)
\[
\begin{align*}
\text{h}_1(“thisisavirus.com”) & \rightarrow 2 \\
\text{h}_2(“thisisavirus.com”) & \rightarrow 1 \\
\text{h}_3(“thisisavirus.com”) & \rightarrow 4
\end{align*}
\]

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Bloom Filters: Contains

function contains(x)
    return \( t_1[h_1(x)] \) \( =\) 1 \& \( t_2[h_2(x)] \) \( =\) 1 \& \( \cdots \) \& \( t_k[h_k(x)] \) \( =\) 1

Returns True if the bit vector for each hash function has bit 1 at index determined by that hash function, otherwise returns False.
Bloom Filters: Example

bloom filter \( t \) with \( m = 5 \) that uses \( k = 3 \) hash functions

contains(“thisisavirus.com”)

function `contains(x)`

return \( t_1[h_1(x)] = 1 \land t_2[h_2(x)] = 1 \land \cdots \land t_k[h_k(x)] = 1 \)

<table>
<thead>
<tr>
<th>Index →</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
**Bloom Filters: Example**

Bloom filter \( t \) of length \( m = 5 \) that uses \( k = 3 \) hash functions

\[
\text{contains(“thisisavirus.com”)}
\]

\[
h_1(“thisisavirus.com”) \rightarrow 2
\]

True

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\text{function} \ \text{contains(x)}
\]

\[
\text{return} \ t_1[h_1(x)] = 1 \land t_2[h_2(x)] = 1 \land \cdots \land t_k[h_k(x)] = 1
\]
Bloom Filters: Example

Bloom filter \( t \) of length \( m = 5 \) that uses \( k = 3 \) hash functions.

Function `contains(x)`

```python
function contains(x):
    return \( t_1[h_1(x)] = 1 \land t_2[h_2(x)] = 1 \land \ldots \land t_k[h_k(x)] = 1 \)
```

**contains(“thisisavirus.com”)**

\( h_1(“thisisavirus.com”) \rightarrow 2 \)

\( h_2(“thisisavirus.com”) \rightarrow 1 \)

```
<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Bloom Filters: Example

Bloom filter \( t \) of length \( m = 5 \) that uses \( k = 3 \) hash functions

Function: \( \text{contains}(x) \)

\[
\text{return } t_1[h_1(x)] \land t_2[h_2(x)] \land \cdots \land t_k[h_k(x)] = 1
\]

\( \text{contains(“thisisavirus.com”)} \)

\[
\begin{align*}
  h_1(“thisisavirus.com”) & \rightarrow 2 \\
  h_2(“thisisavirus.com”) & \rightarrow 1 \\
  h_3(“thisisavirus.com”) & \rightarrow 4
\end{align*}
\]

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Bloom Filters: Example

bloom filter $t$ of length $m = 5$ that uses $k = 3$ hash functions

$\text{contains(“thisisavirus.com”)}$

\[
\text{function } \text{contains}(x) \\
\text{return } t_1[h_1(x)] = 1 \land t_2[h_2(x)] = 1 \land \cdots \land t_k[h_k(x)] = 1
\]

True True True

Since all conditions satisfied, returns True (correctly)

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Bloom Filters: False Positives

bloom filter t of length m = 5 that uses k = 3 hash functions

add("totallynotsuspicious.com")

function ADD(x)
for i = 1, ..., k: do
    \( t_i[h_i(x)] = 1 \)

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Bloom Filters: False Positives

A bloom filter $t$ of length $m = 5$ that uses $k = 3$ hash functions.

```
add("totallynotsuspicious.com")
h_1("totallynotsuspicious.com") \rightarrow 1
```

**Function ADD($x$)**

```
for i = 1, \ldots, k: do  
\quad t_i[h_i(x)] = 1
```

<table>
<thead>
<tr>
<th>Index $\rightarrow$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Bloom Filters: False Positives

bloom filter $t$ of length $m = 5$ that uses $k = 3$ hash functions

**function** $\text{ADD}(x)$

for $i = 1, \ldots, k$: do

$t_i[h_i(x)] = 1$

add(“totallynotsuspicious.com”)

$h_1(“totallynotsuspicious.com”) \rightarrow 1$

$h_2(“totallynotsuspicious.com”) \rightarrow 0$

<table>
<thead>
<tr>
<th>Index $i$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Bloom Filters: False Positives

Bloom filter \( t \) of length \( m = 5 \) that uses \( k = 3 \) hash functions

function `ADD(x)`

for \( i = 1, \ldots, k \): do

\[ t_i[h_i(x)] = 1 \]

add(“totallynotsuspicious.com”)

\[ h_1(“totallynotsuspicious.com”) \to 1 \]

\[ h_2(“totallynotsuspicious.com”) \to 0 \]

\[ h_3(“totallynotsuspicious.com”) \to 4 \]

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Collision, is already set to 1
Bloom Filters: False Positives

bloom filter of length $m = 5$ that uses $k = 3$ hash functions

add(“totallynotsuspicious.com”)

$h_1(“totallynotsuspicious.com”) \rightarrow 1$

$h_2(“totallynotsuspicious.com”) \rightarrow 0$

$h_3(“totallynotsuspicious.com”) \rightarrow 4$

function $\text{ADD}(x)$

for $i = 1, \ldots, k$: do

$t_i[h_i(x)] = 1$

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Bloom Filters: Example

Bloom filter $t$ of length $m = 5$ that uses $k = 3$ hash functions

contains(“verynormalsite.com”)

function contains(x)
    return $t_1[h_1(x)] == 1 \land t_2[h_2(x)] == 1 \land \cdots \land t_k[h_k(x)] == 1$

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Bloom Filters: Example

Bloom filter $t$ of length $m = 5$ that uses $k = 3$ hash functions

contains("verynormalsite.com")

function contains(x)
  return $t_1[h_1(x)] \land t_2[h_2(x)] \land \cdots \land t_k[h_k(x)] = 1$

True

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Bloom Filters: Example

bloom filter \( t \) of length \( m = 5 \) that uses \( k = 3 \) hash functions

contains("verynormalsite.com")

\[
\begin{align*}
& h_1(\text{"verynormalsite.com"}) \rightarrow 2 \\
& h_2(\text{"verynormalsite.com"}) \rightarrow 0
\end{align*}
\]

True True

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

function contains(x)
return \( t_1[h_1(x)] = 1 \land t_2[h_2(x)] = 1 \land \cdots \land t_k[h_k(x)] = 1 \)
Bloom Filters: Example

A bloom filter $t$ of length $m = 5$ that uses $k = 3$ hash functions

**function** `contains(x)`

**return** $t_1[h_1(x)] \land t_2[h_2(x)] \land \cdots \land t_k[h_k(x)] = 1$

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

`contains(`verynormalsite.com`)`

- $h_1(`verynormalsite.com`) \rightarrow 2$
- $h_2(`verynormalsite.com`) \rightarrow 0$
- $h_3(`verynormalsite.com`) \rightarrow 4$
Bloom Filters: Example

bloom filter $t$ of length $m = 5$ that uses $k = 3$ hash functions

contains("verynormalsite.com")
- $h_1("verynormalsite.com") \rightarrow 2$
- $h_2("verynormalsite.com") \rightarrow 0$
- $h_3("verynormalsite.com") \rightarrow 4$

Since all conditions satisfied, returns True (incorrectly)

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Bloom Filters: Summary

- An empty bloom filter is an empty $k \times m$ bit array with all values initialized to zeros
  - $k =$ number of hash functions
  - $m =$ size of each array in the bloom filter
- $\text{add}(x)$ runs in $O(k)$ time
- $\text{contains}(x)$ runs in $O(k)$ time
- Requires $O(km)$ space (in bits!)
- Probability of false positives from collisions can be reduced by increasing the size of the bloom filter
Bloom Filters: Application

- Google Chrome has a database of malicious URLs, but it takes a long time to query.
- Want an in-browser structure, so needs to be efficient and be space-efficient.
- Want it so that can check if a URL is in structure:
  - If return False, then definitely not in the structure (don’t need to do expensive database lookup, website is safe)
  - If return True, the URL may or may not be in the structure. Have to perform expensive lookup in this rare case.
False positive probability
Comparison with Hash tables - Space

- Google storing 5 million URLs, each URL 40 bytes.
- Bloom filter with $k=8$ and $m = 10,000,000$. 
Say avg user visits 100,000 URLs in a year, of which 2,000 are malicious.
0.5 seconds to do lookup in the database, 1ms for lookup in Bloom filter.
Suppose the false positive rate is 2%
Bloom Filters: Many Applications

- Any scenario where space and efficiency are important.
- Used a lot in networking
- In distributed systems when want to check consistency of data across different locations, might send a Bloom filter rather than the full set of data being stored.
- Google BigTable uses Bloom filters to reduce the disk lookups for non-existent rows and columns
- Internet routers often use Bloom filters to track blocked IP addresses.
- And on and on...
Bloom Filters typical example...

of randomized algorithms and randomized data structures.

- Simple
- Fast
- Efficient
- Elegant
- Useful!

- You’ll be implementing Bloom filters on pset 4. Enjoy!