Announcements

• PSet 3 due today
• PSet 2 returned yesterday
• PSet 4 posted this evening
  – Last PSet prior to midterm (midterm is in exactly two weeks from now)
  – Midterm info will follow soon
  – PSet 5 will only come after the midterm in two weeks
**Recap Variance – Properties**

**Definition.** The variance of a (discrete) RV $X$ is

$$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \sum_x p_x(x) \cdot (x - \mathbb{E}[X])^2$$

**Theorem.** For any $a, b \in \mathbb{R}$, $\text{Var}(a \cdot X + b) = a^2 \cdot \text{Var}(X)$

**Theorem.** $\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$
**Variance**

**Theorem.** $\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

**Proof:**

$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$

$= \mathbb{E}[X^2] - 2\mathbb{E}[X] \cdot X + \mathbb{E}[X]^2$

$= \mathbb{E}(X^2) - 2\mathbb{E}[X] \mathbb{E}[X] + \mathbb{E}[X]^2$

$= \mathbb{E}[X^2] - \mathbb{E}[X]^2$  

(linearity of expectation!)

Recall $\mathbb{E}[X]$ is a constant

$\mathbb{E}[X^2]$ and $\mathbb{E}[X]^2$ are different!
Variance of Indicator Random Variables

Suppose that $X_A$ is an indicator RV for event $A$ with $P(A) = p$ so

$$\mathbb{E}[X_A] = P(A) = p$$

Since $X_A$ only takes on values 0 and 1, we always have $X_A^2 = X_A$ so

$$\text{Var}(X_A) = \mathbb{E}[X_A^2] - \mathbb{E}[X_A]^2 = \mathbb{E}[X_A] - \mathbb{E}[X_A]^2 = p - p^2 = p(1 - p)$$
In General, \( \text{Var}(X + Y) \neq \text{Var}(X) + \text{Var}(Y) \)

Proof by counter-example:

• Let \( X \) be a r.v. with pmf \( P(X = 1) = P(X = -1) = 1/2 \)
  – What is \( \mathbb{E}[X] \) and \( \text{Var}(X) \)?

• Let \( Y = -X \)
  – What is \( \mathbb{E}[Y] \) and \( \text{Var}(Y) \)?

What is \( \text{Var}(X + Y) \)?
Agenda

- Variance
- Properties of Variance
- Independent Random Variables
- Properties of Independent Random Variables
- An Application: Bloom Filters!
**Random Variables and Independence**

**Definition.** Two random variables $X, Y$ are (mutually) independent if for all $x, y$,

$$P(X = x, Y = y) = P(X = x) \cdot P(Y = y)$$

**Intuition:** Knowing $X$ doesn’t help you guess $Y$ and vice versa.

**Definition.** The random variables $X_1, \ldots, X_n$ are (mutually) independent if for all $x_1, \ldots, x_n$,

$$P(X_1 = x_1, \ldots, X_n = x_n) = P(X_1 = x_1) \cdots P(X_n = x_n)$$

Note: No need to check for all subsets, but need to check for all outcomes!
Example

Let $X$ be the number of heads in $n$ independent coin flips of the same coin. Let $Y = X \mod 2$ be the parity (even/odd) of $X$. Are $X$ and $Y$ independent?

Poll:
pollev.com/paulbeame028

A. Yes
B. No
Example

Make $2n$ independent coin flips of the same coin. Let $X$ be the number of heads in the first $n$ flips and $Y$ be the number of heads in the last $n$ flips. Are $X$ and $Y$ independent?

Poll: pollev.com/paulbeame028

A. Yes
B. No
Agenda

• Variance
• Properties of Variance
• Independent Random Variables
• Properties of Independent Random Variables
• An Application: Bloom Filters!
Important Facts about Independent Random Variables

**Theorem.** If $X, Y$ independent, $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$

**Theorem.** If $X, Y$ independent, $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$

**Corollary.** If $X_1, X_2, \ldots, X_n$ mutually independent,

$$\text{Var}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i}^{n} \text{Var}(X_i)$$
Example – Coin Tosses

We flip \( n \) independent coins, each one heads with probability \( p \)

- \( X_i = \begin{cases} 1, & \text{\( i \)th outcome is heads} \\ 0, & \text{\( i \)th outcome is tails.} \end{cases} \)

- \( Z = \) number of heads

What is \( \mathbb{E}[Z] \)? What is \( \text{Var}(Z) \)?

Note: \( X_1, \ldots, X_n \) are mutually independent! [Verify it formally!]

\[
\text{Var}(Z) = \sum_{i=1}^{n} \text{Var}(X_i) = n \cdot p(1 - p)
\]

Note \( \text{Var}(X_i) = p(1 - p) \)

Fact. \( Z = \sum_{i=1}^{n} X_i \)

\[
P(X_i = 1) = p \\
P(X_i = 0) = 1 - p
\]

\[
P(Z = k) = \binom{n}{k} p^k (1 - p)^{n-k}
\]
(Not Covered) Proof of $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$

**Theorem.** If $X, Y$ independent, $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$

**Proof**

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

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

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

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

\[
= \mathbb{E}[X] \cdot \mathbb{E}[Y]
\]

Note: NOT true in general; see earlier example $\mathbb{E}[X^2] \neq \mathbb{E}[X]^2$
Proof of $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$

**Theorem.** If $X, Y$ independent, $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$

**Proof**

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

By linearity,

\[
= \mathbb{E}[X^2 + 2XY + Y^2] - (\mathbb{E}[X] + \mathbb{E}[Y])^2
\]

\[
= \mathbb{E}[X^2] + 2 \mathbb{E}[XY] + \mathbb{E}[Y^2] - (\mathbb{E}[X]^2 + 2 \mathbb{E}[X] \mathbb{E}[Y] + \mathbb{E}[Y]^2)
\]

\[
= \mathbb{E}[X^2] - \mathbb{E}[X]^2 + \mathbb{E}[Y^2] - \mathbb{E}[Y]^2 + 2 \mathbb{E}[XY] - 2 \mathbb{E}[X] \mathbb{E}[Y]
\]

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

\[
= \text{Var}(X) + \text{Var}(Y)
\]

Equal by independence.
Brain Break
Agenda

• Variance
• Properties of Variance
• Independent Random Variables
• Properties of Independent Random Variables
• An Application: Bloom Filters!
Basic Problem

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

**Example.** $U = \text{set of 128 bit strings}$

$S = \text{subset of strings of interest}$

| $|U|$ | $≈ 2^{128}$ |
| $|S|$ | $≈ 1000$ |

**Two goals:**

1. **Very fast** (ideally constant time) answers to queries “Is $x \in S$?” for any $x \in U$.
2. **Minimal storage** requirements.
Naïve Solution I – Constant Time

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

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

$$A[x] = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases}$$

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

→ constant time! 😊 😁

**Storage:** Require storing $2^{128}$ bits, even for small $S$. 😞 😞
Naïve Solution II – 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 $A$ of size $m$ using a hash function $h$.

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

**Storage:** $m$ elements (size of array)

hash function $h: U \rightarrow [m]$
**Hash Table**

**Idea:** Map elements in $S$ into an array $A$ of size $m$ using a hash function $h$.

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

**Storage:** $m$ elements (size of array)

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

**Challenge 2:** Ensure $m = O(|S|)$.
Hashing: collisions

Collisions occur when \( h(x) = h(y) \) for some distinct \( x, y \in S \), i.e., two elements of set map to the same location.

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

\[ Z = Z \mod n \]
Good hash functions to keep collisions low

- The hash function $h$ is good iff it
  - distributes elements uniformly across the $m$ array locations so that
  - pairs of elements are mapped independently

“Universal Hash Functions” – see CSE 332
Hashing: summary

**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, i.e., \( m = \Omega(|S|) \)

In some cases, \(|S|\) is huge, or not known a-priori ...

Can we do better!?
Bloom Filters
to the rescue
(Named after Burton Howard Bloom)
Bloom Filters – Main points

- Probabilistic data structure.
- Close cousins of hash tables.
  - But: Ridiculously space efficient
- Occasional errors, specifically false positives.
Bloom Filters

- Stores information about a set of elements $S \subseteq U$.
- Supports two operations:
  1. $\text{add}(x)$ - adds $x \in U$ to the set $S$
  2. $\text{contains}(x)$ – ideally: true if $x \in S$, false otherwise

Instead, relaxed guarantees:
- False $\rightarrow$ definitely not in $S$
- True $\rightarrow$ possibly in $S$
  [i.e. we could have false positives]
Bloom Filters – Why Accept False Positives?

• **Speed** – both **add** and **contains** 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!
• **Fallback mechanism** – can distinguish false positives from true positives with extra cost
  – Ok if mostly negatives expected + low false positive rate
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.
Bloom Filters – More 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 disk lookups
- Internet routers often use Bloom filters to track blocked IP addresses.
- And on and on...
What you can’t do with Bloom filters

• There is no delete operation
  – Once you have added something to a Bloom filter for $S$, it stays

• You can’t use a Bloom filter to name any element of $S$

But what you can do makes them very effective!
Bloom Filters – Ingredients

Basic data structure is a $k \times m$ binary array “the Bloom filter”

- $k$ rows $t_1, \ldots, t_k$, each of size $m$
- Think of each row as an $m$-bit vector

$k$ different hash functions $h_1, \ldots, h_k : U \rightarrow [m]$
Bloom Filters – Three operations

• Set up Bloom filter for $S = \emptyset$

• Update Bloom filter for $S \leftarrow S \cup \{x\}$

• Check if $x \in S$

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

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

function $\text{CONTAINS}(x)$
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 - Initialization

function INITIALIZE($k, m$)
  for $i = 1, \ldots, k$: do
    $t_i$ = new bit vector of $m$ 0s

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

\begin{function}
\text{INITIALIZE}(k, m)
\begin{align*}
\text{for } i = 1, \ldots, k: \text{ do} \\
& t_i = \text{new bit vector of } m \text{ 0s}
\end{align*}
\end{function}

<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>
function \text{ADD}(x)

\text{for } i = 1, \ldots, k: \text{ do}

\text{for each hash function } h_i \text{ result of hash function } h_i(x) \text{ on } x

\text{Index into } i\text{-th bit-vector, at index produced by hash function and set to 1}
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

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

\[
\begin{array}{c|c|c|c|c|c|c}
\text{Index} & 0 & 1 & 2 & 3 & 4 \\
\hline
\text{t}_1 & 0 & 0 & 0 & 0 & 0 \\
\text{t}_2 & 0 & 0 & 0 & 0 & 0 \\
\text{t}_3 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

add(“thisisavirus.com”)

\[ h_1(“thisisavirus.com”) \rightarrow 2 \]
Bloom Filters: Example

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

**function ADD($x$)**

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

t$_i[h_i(x)] = 1$

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>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** `ADD(x)`

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

add(“thisisavirus.com”)

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

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

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

<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>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

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

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

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

add(“thisisavirus.com”)

\( h_1(“thisisavirus.com”) \rightarrow 2 \)
\( h_2(“thisisavirus.com”) \rightarrow 1 \)
\( h_3(“thisisavirus.com”) \rightarrow 4 \)

<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>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 $t_1[h_1(x)] == 1 \land t_2[h_2(x)] == 1 \land \ldots \land t_k[h_k(x)] == 1$

Returns True if the bit vector $t_i$ for each hash function has bit 1 at index determined by $h_i(x)$, Returns False otherwise
Bloom Filters: Example

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

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

contains(“thisisavirus.com”)

<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

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

contains(“thisisavirus.com”)

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

Table:

<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>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

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

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** \( \text{CONTAINS}(x) \)

```python
return t_1[h_1(x)] == 1 ∧ t_2[h_2(x)] == 1 ∧ ⋯ ∧ t_k[h_k(x)] == 1
```

Index $→$ 0 1 2 3 4

<table>
<thead>
<tr>
<th></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>

contains(“thisisavirus.com”)

- $h_1(“thisisavirus.com”) → 2$
- $h_2(“thisisavirus.com”) → 1$
- $h_3(“thisisavirus.com”) → 4$
Bloom Filters: Example

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

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

contains(“thisisavirus.com”)

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

Since all conditions satisfied, returns \text{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

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

add("totallynotsuspicious.com")

<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

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

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

$t_i[h_i(x)] = 1$

add(“totallynotsuspicious.com”)

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

<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>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

**function** `ADD(x)`

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

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

add(“totallynotsuspicous.com”)

\[ h_1(“totallynotsuspicous.com”) \rightarrow 1 \]

\[ h_2(“totallynotsuspicous.com”) \rightarrow 0 \]

<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>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”)

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

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

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

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

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

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

add(“totallynotsuspicious.com”)

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

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

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

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

contains(“verynormalsite.com”)

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

contains(“verynormalsite.com”)

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

<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>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

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

contains(“verynormalsite.com”) contains(“verynormalsite.com”) → 2
$h_1(“verynormalsite.com”) → 2$
h$_2(“verynormalsite.com”) → 0

<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: False Positives**

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

```python
function contains(x):
    return t_1[h_1(x)] == 1 ∧ t_2[h_2(x)] == 1 ∧ ⋯ ∧ t_k[h_k(x)] == 1

contains("verynormalsite.com")
```

- $h_1("verynormalsite.com") → 2$
- $h_2("verynormalsite.com") → 0$
- $h_3("verynormalsite.com") → 4$

<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>
Bloom Filters: False Positives

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

```python
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
```

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)**
Analysis: False positive probability

**Question:** For an element $x \in U$, what is the probability that $\text{contains}(x)$ returns true if $\text{add}(x)$ was never executed before?

Probability over what?! Over the choice of the $h_1, \ldots, h_k$

Assumptions for the analysis (somewhat stronger than for ordinary hashing):

- Each $h_i(x)$ is uniformly distributed in $[m]$ for all $x$ and $i$
- Hash function outputs for each $h_i$ are mutually independent (not just in pairs)
- Different hash functions are independent of each other
False positive probability – Events

Assume we perform \( \text{add}(x_1), \ldots, \text{add}(x_n) \)
\[ + \text{contains}(x) \text{ for } x \notin \{x_1, \ldots, x_n\} \]

Event \( E_i \) holds iff \( h_i(x) \in \{h_i(x_1), \ldots, h_i(x_n)\} \)

\[
P(\text{false positive}) = P(E_1 \cap E_2 \cap \cdots \cap E_k) = \prod_{i=1}^{k} P(E_i)
\]

\( h_1, \ldots, h_k \) independent
False positive probability – Events

Event $E_i$ holds iff $h_i(x) \in \{h_i(x_1), ..., h_i(x_n)\}$

Event $E_i^c$ holds iff $h_i(x) \neq h_i(x_1)$ and ... and $h_i(x) \neq h_i(x_n)$

$$P(E_i^c) = \sum_{z=1}^{m} P(h_i(x) = z) \cdot P(E_i^c \mid h_i(x) = z)$$

LTP
False positive probability – Events

Event $E_i^c$ holds iff $h_i(x) \neq h_i(x_1)$ and ...
and $h_i(x) \neq h_i(x_n)$

\[
P(E_i^c | h_i(x) = z) = P(h_i(x_1) \neq z, ..., h_i(x_n) \neq z | h_i(x) = z)
\]

\[
= P(h_i(x_1) \neq z, ..., h_i(x_n) \neq z)
\]

\[
= \prod_{j=1}^{n} P(h_i(x_j) \neq z)
\]

\[
= \prod_{j=1}^{n} \left(1 - \frac{1}{m}\right) = \left(1 - \frac{1}{m}\right)^n
\]

\[
P(E_i^c) = \sum_{z=1}^{m} P(h_i(x) = z) \cdot P(E_i^c | h_i(x) = z) = \left(1 - \frac{1}{m}\right)^n
\]
False positive probability – Events

Event $E_i$ holds iff $h_i(x) \in \{h_i(x_1), \ldots, h_i(x_n)\}$

Event $E_i^c$ holds iff $h_i(x) \neq h_i(x_1)$ and \ldots and $h_i(x) \neq h_i(x_n)$

$$P(E_i^c) = \left(1 - \frac{1}{m}\right)^n$$

$$\text{FPR} = \prod_{i=1}^{k} \left(1 - P(E_i^c)\right) = \left(1 - \left(1 - \frac{1}{m}\right)^n\right)^k$$
False Positivity Rate – Example

\[
\text{FPR} = \left( 1 - \left( 1 - \frac{1}{m} \right)^n \right)^k
\]

e.g., \( n = 5,000,000 \)
\( k = 30 \)
\( m = 2,500,000 \)

\[
\text{FPR} = 1.28\%
\]
Comparison with Hash Tables - **Space**

- Google storing 5 million URLs, each URL 40 bytes.
- Bloom filter with $k = 30$ and $m = 2,500,000$

<table>
<thead>
<tr>
<th>Hash Table</th>
<th>Bloom Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>(optimistic) $5,000,000 \times 40B = 200\text{MB}$</td>
<td>$2,500,000 \times 30 = 75,000,000\text{ bits}$ $&lt; 10\text{ MB}$</td>
</tr>
</tbody>
</table>
Time

- Say avg user visits **102,000** 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 **3%**

\[
\text{false positives} = 100000 \times 0.03 \times 500\text{ms} + 2000 \times 500\text{ms} \approx 25.51\text{ms}
\]

Bloom filter lookup

0.5 seconds DB lookup

malicious URLs

total URLs

\[
1\text{ms} + \frac{100000 \times 0.03 \times 500\text{ms} + 2000 \times 500\text{ms}}{102000} \approx 25.51\text{ms}
\]
Bloom Filters typical of....

... randomized algorithms and randomized data structures.

• Simple
• Fast
• Efficient
• Elegant
• Useful!