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

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

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 $a, b \in \mathbb{R}$,

$$\text{Var}(aX + b) = a^2 \text{Var}(X)$$

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

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

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 $X$. More Useful
\[ E(X_i) = \frac{1}{2} \]

\[
\begin{array}{ccc}
X_i & P_r(X_i = x) & X^2_i \\
0 & \frac{1}{2} & 0 \\
1 & \frac{1}{2} & 1 \\
\end{array}
\]

\[
\frac{Y}{(X_i - E(X_i))^2} = \frac{4}{9}
\]

\[
V_{an}(X_i) = E\left[\left(X_i - E(X_i)\right)^2\right] = \frac{1}{4}
\]

\[
= E\left[X_i^2\right] - \left[E(X_i)\right]^2 = \frac{1}{2} - \left(\frac{1}{2}\right)^2 = \frac{1}{4}
\]
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.

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

$$\equiv \forall x \ P(X=x | E) = P(X=x) \quad [P(E)>0]$$

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.

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

$$\equiv \forall x, y \ P(X=x | Y=y) = P(X=x) \quad [P(Y=y)>0]$$

Intuition as before: knowing $X$ doesn’t help you guess $Y$ or $E$ and vice versa.
**Important facts about independent random variables**

Theorem: If $X$ & $Y$ are independent, then $\mathbb{E}[X \cdot Y] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$.

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 $\text{Var}[X_1 + X_2 + \ldots + X_n] = \text{Var}[X_1] + \text{Var}[X_2] + \ldots + \text{Var}[X_n]$. 
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.

\[ W = n X_1 \]

\[ = x_1 + x_2 + x_3 + \ldots + x_n \]

\[ \text{versus} \]

\[ E(Z) = \frac{1}{2} \]

\[ E(3) = 3 \times \frac{1}{2} + 0 \times \frac{1}{2} = \frac{3}{2} \]

\[ E(W) = n \times E(X_1) = n \times \frac{1}{2} = \frac{n}{2} \]

\[ E[(W - E(W))^2] = \frac{3}{12} \]
\begin{align*}
Z &= X_1 + X_2 + \cdots + X_n \\
W &= nX_1
\end{align*}

\begin{align*}
\text{Var}(X_1) &= \frac{1}{4} \\
W &= \begin{cases} 
\frac{n}{2} & \text{if } n \text{ is even} \\
0 & \text{if } n \text{ is odd}
\end{cases}
\end{align*}

\begin{align*}
\text{Var}(W) &= \mathbb{E}(W^2) - \left( \mathbb{E}(W) \right)^2 \\
&= \frac{n^3}{2} + \frac{n^3}{2} - \left( \frac{n}{2} \right)^2 = \frac{n^3}{2} - \frac{n^3}{4} = \frac{n^3}{4}
\end{align*}

\begin{align*}
\text{Var}(Z) &= \sum_{i=1}^{n} \text{Var}(X_i) \\
\text{Var}(W) &= \text{Var}(W)
\end{align*}

\text{by independence}

\text{for } W \text{ compute directly}

\begin{align*}
\begin{array}{c|c|c|c}
X_1 & X_2 & X_3 & X_4 \\
\hline
0 & 0 & 1 & 2 \\
0 & 1 & 0 & 2 \\
1 & 0 & 0 & 2 \\
1 & 1 & 2 & 2
\end{array}
\end{align*}
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 \( E[XY] = E[X] \cdot E[Y] \)
- **Proof:**

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

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

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

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

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

**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:

\[
\begin{align*}
\text{Var}(X + Y) &= E[(X + Y)^2] - (E[X + Y])^2 \\
&= E[X^2 + 2XY + Y^2] - (E[X] + E[Y])^2 \\
&= \text{Var}(X) + \text{Var}(Y) + 2(E[X]E[Y] - E[X]E[Y]) \\
&= \text{Var}(X) + \text{Var}(Y)
\end{align*}
\]
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. $U = \text{set of 128 bit strings}$
$S = \text{subset of strings of interest}$

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.

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

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

→ constant time! 😊 😊

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

![Diagram of hash table]

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

**Idea:** Map elements in $S$ into an array

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

**Storage:** $m$ elements

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

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

Chaining

Example:

- $S = \{w_1, w_2, \ldots, w_5\}$
- $A$
- $h(w_1) = 3$
- $h(w_2) = 2$
- $h(w_3) = n - 1$
- $h(w_4) = 3$
- $h(w_5) = 3$

Storage $m$ elements
Hashing — Collisions

- **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!

  Analyze hashing: Assume hash fn maps each elt $x \in U$ independently to uniformly random location in table

  Reasonable if $U$ properly large
**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
- 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

function INITIALIZE \((k, m)\)

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

\[
t_i = \text{new bit vector of } m \text{ 0's}
\]
Bloom Filters: Example

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

**function** `INITIALIZE(k, m)`

```plaintext
for $i = 1, \ldots, k$: do
  $t_i = $ new bit vector of $m$ 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**

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

h_i(x) \rightarrow \text{result of hash function } h_i \text{ on } x
```

for each hash function $h_i$
Bloom Filters: Add

**function** ADD(x)

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

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

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

for each hash function $h_i$
Bloom Filters: Example

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

**function** `ADD(x)`

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

add(“thisisavirus.com”)

<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: Example

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

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

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

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

Example:

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

<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

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

add(“thisisavirus.com”)

\[ h_1(“thisisavirus.com”) \rightarrow 2 \]
\[ h_2(“thisisavirus.com”) \rightarrow 1 \]

<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><strong>1</strong></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td><strong>1</strong></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</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 $t_1[h_1(x)] = 1 \land t_2[h_2(x)] = 1 \land \cdots \land t_k[h_k(x)] = 1$

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

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

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

<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

contains("thisisavirus.com")

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

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>
Bloom Filters: Example

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

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

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

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

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

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

<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

**Function** `contains(x)`

```python
function contains(x):
    return t1[h1(x)] == 1 \land t2[h2(x)] == 1 \land \ldots \land t_k[h_k(x)] == 1
```

### Contains("thisisavirus.com")

- $h_1("thisisavirus.com") \rightarrow 2$
- $h_2("thisisavirus.com") \rightarrow 1$
- $h_3("thisisavirus.com") \rightarrow 4$

### Table

<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

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>

Since all conditions satisfied, returns True (correctly)
Bloom Filters: False Positives

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

\[
\text{add(“totallynotsuspicious.com”)}
\]

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

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

<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>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

$t_i[h_i(x)] = 1$

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

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

add("totallynotsuspicious.com")

\[
h_1("totallynotsuspicious.com") \rightarrow 1
\]

\[
h_2("totallynotsuspicious.com") \rightarrow 0
\]

<table>
<thead>
<tr>
<th>Index</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 \( i = 1, \ldots, k \)
\( t_i[h_i(x)] = 1 \)

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

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

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

add(“totallynotsuspicious.com”)

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

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

$h_3(“totallynotsuspicious.com”) \rightarrow 4$
Bloom Filters: Example

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

\[
\text{contains("verynormalsite.com")}
\]

function \text{contains}(x)

\text{return } t_1[h_1(x)] \land t_2[h_2(x)] \land t_3[h_3(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")

$h_1("verynormalsite.com") \rightarrow 2$

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>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**

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

**Example:**

**contains(“verynormalsite.com”)**

\[ h_1(“verynormalsite.com”) \rightarrow 2 \]
\[ h_2(“verynormalsite.com”) \rightarrow 0 \]

**Function contains(\( x \))**

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

<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

**function** `contains(x)`

```plaintext
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>

```

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.

```plaintext
contains("verynormalsite.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>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>
```

Since all conditions satisfied, returns True (incorrectly)

\( m \) elts.

\( m > n \)

\( m = 8n \)
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
- \texttt{add(x)} runs in \( O(k) \) time
- \texttt{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**

Suppose a new URL arrives, I'm storing.

Assumption: hashes are completely random.
**Comparison with Hash tables - Space**

- Google storing 5 million URLs, each URL 40 bytes.
- Bloom filter with $k=8$ and $m = 10,000,000$. 

<table>
<thead>
<tr>
<th>Hash Table</th>
<th>Bloom Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison with Hash tables - Time

- 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!