CSE 312
Foundations of Computing II

Lecture 13: Wrap up Poisson r.v.s + Bloom Filters

Anna’s office hours on Saturday (tmw) from 2-3pm
(not 3-4pm)
Agenda

- More on Poisson random variables
- An Application: Bloom Filters!
Preview: Poisson

Model: \( X \) is \# events that occur in an hour

– Expect to see 3 events per hour (but will be random)
– The expected number of events in \( t \) hours, is \( 3t \)
– Occurrence of events on disjoint time intervals is independent

Example – Modelling car arrivals at an intersection

\( X = \# \) of cars passing through a light in 1 hour
Example – Model the process of cars passing through a light in 1 hour

\( X = \# \) cars passing through a light in 1 hour. Disjoint time intervals are independent.

Know: \( \mathbb{E}[X] = \lambda \) for some given \( \lambda > 0 \)

Discrete version: \( n \) intervals, each of length \( 1/n \).

In each interval, there is a car with probability \( p = \lambda/n \) (assume \( \leq 1 \) car can pass by)

Each interval is Bernoulli: \( X_i = 1 \) if car in \( i^{th} \) interval (0 otherwise). \( P(X_i = 1) = \lambda / n \)

\[
X = \sum_{i=1}^{n} X_i \quad X \sim \text{Bin}(n, p) \quad P(X = i) = \binom{n}{i} \left( \frac{\lambda}{n} \right)^i \left( 1 - \frac{\lambda}{n} \right)^{n-i}
\]

indeed! \( \mathbb{E}[X] = pn = \lambda \)
Don’t like discretization

We want now \( n \to \infty \)

\[
P(X = i) = \binom{n}{i} \left( \frac{\lambda}{n} \right)^i \left( 1 - \frac{\lambda}{n} \right)^{n-i}
\]

\[
\to P(X = i) = e^{-\lambda} \cdot \frac{\lambda^i}{i!}
\]
Poisson Distribution

- Suppose “events” happen, independently, at an average rate of $\lambda$ per unit time.
- Let $X$ be the actual number of events happening in a given time unit. Then $X$ is a Poisson r.v. with parameter $\lambda$ (denoted $X \sim \text{Poi}(\lambda)$) and has distribution (PMF):

$$P(X = i) = e^{-\lambda} \cdot \frac{\lambda^i}{i!} \quad i = 0, 1, 2, ...$$

Several examples of “Poisson processes”:
- # of cars passing through a traffic light in 1 hour
- # of requests to web servers in an hour
- # of photons hitting a light detector in a given interval
- # of patients arriving to ER within an hour

Assume fixed average rate

$$E(X) = \lambda$$
$$\text{Var}(X) = \lambda$$
Poisson Random Variables

**Definition.** A Poisson random variable $X$ with parameter $\lambda \geq 0$ is such that for all $i = 0, 1, 2, 3, \ldots$,

$$P(X = i) = e^{-\lambda} \cdot \frac{\lambda^i}{i!}$$

Poisson approximates binomial when:
- $n$ is very large, $p$ is very small, and $\lambda = np$ is “moderate”
  - e.g. $(n > 20$ and $p < 0.05)$, $(n > 100$ and $p < 0.1)$
- Formally, Binomial approaches Poisson in the limit as $n \to \infty$ (equivalently, $p \to 0$) while holding $np = \lambda$
\( \lambda = 5 \)
\( p = \frac{5}{n} \)
\( n = 10, 15, 20 \)

\( as \ n \to \infty, \ \text{Binomial}(n, \ p = \frac{\lambda}{n}) \to \text{pois}(\lambda) \)
Sum of Independent Poisson RVs

Let $X \sim \text{Poi}(\lambda_1)$ and $Y \sim \text{Poi}(\lambda_2)$ such that $\lambda = \lambda_1 + \lambda_2$.
Let $Z = X + Y$. What kind of random variable is $Z$?
Aka what is the “distribution” of $Z$?

Intuition first:

• $X$ is measuring number of (type 1) events that happen in, say, an hour if they happen at an average rate of $\lambda_1$ per hour.
• $Y$ is measuring number of (type 2) events that happen in, say, an hour if they happen at an average rate of $\lambda_2$ per hour.
• $Z$ is measuring total number of events of both types that happen in, say, an hour, if type 1 and type 2 events occur independently.
**Theorem.** Let $X \sim \text{Poi}(\lambda_1)$ and $Y \sim \text{Poi}(\lambda_2)$ such that $\lambda = \lambda_1 + \lambda_2$.

Let $Z = X + Y$. For all $z = 0, 1, 2, 3 ...,$

$$P(Z = z) = e^{-\lambda} \cdot \frac{\lambda^z}{z!}$$

More generally, let $X_1 \sim \text{Poi}(\lambda_1), \ldots, X_n \sim \text{Poi}(\lambda_n)$ such that $\lambda = \sum_i \lambda_i$.

Let $Z = \sum_i X_i$

$$P(Z = z) = e^{-\lambda} \cdot \frac{\lambda^z}{z!}$$
**Theorem.** Let $X \sim \text{Poi}(\lambda_1)$ and $Y \sim \text{Poi}(\lambda_2)$ such that $\lambda = \lambda_1 + \lambda_2$.
Let $Z = X + Y$. For all $z = 0,1,2,3 \ldots$,

$$P(Z = z) = e^{-\lambda} \cdot \frac{\lambda^z}{z!} \quad z = 0,1,2,3 \ldots$$

**Proof**

$$P(Z = z) = \sum_{j=0}^{z} P(X = j, Y = z - j)$$

Law of total probability

$$(\alpha + \beta)^2 = \sum_{j=0}^{2} \binom{2}{j} \alpha^j \beta^{2-j}$$
Proof

\[ P(Z = z) = \sum_{j=0}^{z} P(X = j, Y = z-j) \]

\[ = \sum_{j=0}^{z} P(X = j) P(Y = z-j) = \sum_{j=0}^{z} e^{-\lambda_1} \frac{\lambda_1^j}{j!} e^{-\lambda_2} \frac{\lambda_2^{z-j}}{z-j!} \]

\[ = e^{-\lambda_1-\lambda_2} \left( \sum_{j=0}^{z} \frac{1}{j! (z-j)!} \lambda_1^j \lambda_2^{z-j} \right) \]

\[ = e^{-\lambda} \left( \sum_{j=0}^{z} \frac{z!}{j! (z-j)!} \frac{\lambda_1^j \lambda_2^{z-j}}{z!} \right) \frac{1}{z!} \]

\[ = e^{-\lambda} \cdot (\lambda_1 + \lambda_2)^z \cdot \frac{1}{z!} = e^{-\lambda} \cdot \lambda^z \cdot \frac{1}{z!} \]

Law of total probability

Independence

Binomial Theorem

\[ n = \lambda_1 + \lambda_2 \]
Poisson Random Variables

**Definition.** A Poisson random variable $X$ with parameter $\lambda \geq 0$ is such that for all $i = 0, 1, 2, 3 \ldots$,

$$P(X = i) = e^{-\lambda} \cdot \frac{\lambda^i}{i!}$$

**General principle:**

- Events happen at an average rate of $\lambda$ per time unit
- Number of events happening at a time unit $X$ is distributed according to $\text{Poi}(\lambda)$
- Poisson approximates Binomial when $n$ is large, $p$ is small, and $np$ is moderate
- Sum of independent Poisson is still a Poisson
Agenda

• Wrap up Poisson random variables
• An Application: Bloom Filters!
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

<table>
<thead>
<tr>
<th>$U$</th>
<th>$\approx 2^{128}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$\approx 1000$</td>
</tr>
</tbody>
</table>

**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.

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>K</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

**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)
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.
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 2:** Ensure \( m = O(|S|) \)

**Challenge 1:** Ensure \( h(x) \neq h(y) \) for most \( x, y \in S \)
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., \( m = \Omega(|S|) \)

\[
X: \# \text{elts that map to location 1 in table} \nonumber
\]
\[
|S| = m \text{ elts} \quad \text{table size} = m 
\]

\[
E(X) = 1 
\]

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
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
• Internet routers often use Bloom filters to track blocked IP addresses.
• 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
• And on and on...
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 - Initialization

\begin{itemize}
  \item Number of hash functions
  \item Size of array associated to each hash function.
\end{itemize}

\begin{itemize}
  \item for each hash function, initialize an empty bit vector of size \( m \)
\end{itemize}

\textbf{function} \texttt{INITIALIZE}(k, m)

\begin{itemize}
  \item \texttt{for} \( i = 1, \ldots, k \): \texttt{do}
  \item \( t_i = \text{new bit vector of} \ m \ 0\text{s} \)
\end{itemize}
Bloom Filters: Example

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

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

for $i = 1, ..., k$: do
  
  $t_i$ = new bit vector of $m$ 0s

<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 \text{ADD}(x)

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

t_i[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

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

$h_1(\text{"thisisavirus.com"}) \rightarrow 2$

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

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

$t_i[h_i(x)] = 1$

### add("thisisavirus.com")

$h_1(\text{"thisisavirus.com"}) \rightarrow 2$

$h_2(\text{"thisisavirus.com"}) \rightarrow 1$

<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

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

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

<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>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")

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

<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

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

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

```
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 $\rightarrow$</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

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

\[
\text{contains}(“thisisavirus.com”) \\
h_1(“thisisavirus.com”) \rightarrow 2
\]

True

<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>( \boxed{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>( \boxed{1} )</td>
</tr>
</tbody>
</table>
Bloom Filters: Example

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("thisisavirus.com")

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

$h_2("thisisavirus.com") \rightarrow 1$

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>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 \textsc{contains}(x)
\hspace{1em} \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{tabular}{|c|c|c|c|c|c|}
\hline
Index & 0 & 1 & 2 & 3 & 4 \\
\hline
\hline
$t_1$ & 0 & 0 & 1 & 0 & 0 \\
\hline
$t_2$ & 0 & 1 & 0 & 0 & 0 \\
\hline
$t_3$ & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{tabular}

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

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

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

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

Since all conditions satisfied, returns True (correctly)

contains(“thisisavirus.com”)

\[
\begin{array}{l}
h_1(“thisisavirus.com”) \rightarrow 2 \\
h_2(“thisisavirus.com”) \rightarrow 1 \\
h_3(“thisisavirus.com”) \rightarrow 4 \\
\end{array}
\]
Bloom Filters: False Positives

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

```
add("totallynotsuspicious.com")
```

**function** ADD($x$)

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

---

<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

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</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")

$h_1("totallynotsuspicious.com") \rightarrow 1$

$h_2("totallynotsuspicious.com") \rightarrow 0$

<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”)

$h_1(“totallynotsuspicious.com”) \rightarrow 1$
$h_2(“totallynotsuspicious.com”) \rightarrow 0$
$h_3(“totallynotsuspicious.com”) \rightarrow 4$

<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 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></th>
<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></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_2$</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$t_3$</td>
<td></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 $\text{CONTAINS}(x)$
  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”)

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

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: 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 $\text{CONTAINS}(x)$

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

contains(“verynormalsite.com”)

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

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

$\begin{array}{c}
h_1(“verynormalsite.com”) \rightarrow 2 \\
h_2(“verynormalsite.com”) \rightarrow 0 \\
h_3(“verynormalsite.com”) \rightarrow 4 \\
\end{array}$

<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”)  

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

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

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

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 – Three operations

- Set up Bloom filter for $S = \emptyset$
  
  function \text{INITIALIZE}(k, m) 
  for $i = 1, \ldots, k$: do 
  $t_i = \text{new bit vector of } m \text{ 0s}$

- Update Bloom filter for $S \leftarrow S \cup \{x\}$
  
  function \text{ADD}(x) 
  for $i = 1, \ldots, k$: do 
  $t_i[h_i(x)] = 1$

- Check if $x \in S$
  
  function \text{CONTAINS}(x) 
  return $t_1[h_1(x)] = 1 \land t_2[h_2(x)] = 1 \land \cdots \land t_k[h_k(x)] = 1$
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!
Brain Break
Analysis: False positive probability

Question: For an element $x \in U$, what is the probability that \texttt{contains}(x) returns true if \texttt{add}(x) was never executed before?
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:
- 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 \( \text{add}(x_1), ..., \text{add}(x_n) \) + \( \text{contains}(x) \) for \( x \notin \{x_1, ..., x_n\} \)

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

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

\( h_1, ..., h_k \) 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 $h_i(x) \neq h_i(x_n)$

$$P(E_i^c) = \sum_{z=1}^{m} P(h_i(x) = z) \cdot P(E_i^c \mid h_i(x) = z)$$
False positive probability – Events

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

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

Independence of values of $h_i$ on different inputs

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

$$= \prod_{j=1}^{n} P(h_i(x_j) \neq z)$$
False positive probability – Events

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

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

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

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

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

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

Independence of values of $h_i$ on different inputs

Outputs of $h_i$ uniformly spread
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 ... and $h_i(x) \neq h_i(x_n)$

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

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

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

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

\[
FPR = 1.28\%
\]
### 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>$2,500,000 \times 30 = 75,000,000$ 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%

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

... randomized algorithms and randomized data structures.

• Simple
• Fast
• Efficient
• Elegant
• Useful!