Last Class:
• Linearity of Expectation

Today:
• An application: Bloom Filters!
• LOTUS
Basic Problem

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

Example. $U = \text{set of 128 bit strings}$
$S = \text{subset of strings of interest}$

$|U| \approx 2^{128}$
$|S| \approx 1000$

Two goals:
1. Very fast (ideally constant time) answers to queries “Is $x \in S$?”
2. Minimal storage requirements.
Naïve Solution – Constant Time

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

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

$$A[x] = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases}$$
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 $2^{128}$ bits, even for small $S$. 😞 😞

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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>K</th>
<th>...</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
**Naïve Solution – Small Storage**

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

$S = \{0, 2, ..., K\}$
Naïve Solution – Small Storage

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

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

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 $n$ using a hash function $h : U \rightarrow \{0,1,\ldots,n-1\}$. 

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

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

$$|S| = n$$

**Hash function** $h : [U] \rightarrow [n]$
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

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

**Challenge 2:** Ensure $n = O(|S|)$
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!
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
- 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.

Example:
- **Network routers**: want to track source IP addresses of certain packets, e.g., blocked IP addresses.
Bloom Filters to the rescue
Bloom Filters: motivation (3)

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

    - If returns false, **definitely** not in bloom filter.
    - If returns 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 \text{INITIALIZE}(k,m)
for \( i = 1, \ldots, k \): do
\( t_i = \text{new bit vector of } m \text{ 0's} \)

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

```plaintext
function INITIALIZE(k, m)
  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

**function** `ADD(x)`

```plaintext
for i = 1, ..., k: 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

\( h_i(x) \to \) result of hash function \( h_i \) on \( x \)
Bloom Filters: Example

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

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

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

\[h_1(\text{"thisisaviru.com"}) \rightarrow 2\]

<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>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")

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

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

<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>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")

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

<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

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

**function** ADD(X)

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

<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>0</td>
<td>\textcolor{red}{1}</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>\textcolor{red}{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>\	extcolor{red}{1}</td>
</tr>
</tbody>
</table>
Bloom Filters: Example

A bloom filter \( t \) with \( m = 5 \) that uses \( k = 3 \) hash functions contains "thisisavirus.com"

```python
function contains(x):
    return t1[h1(x)] == 1 ∧ t2[h2(x)] == 1 ∧ ... ∧ tk[hk(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

**Example**: Bloom filter $t$ of length $m = 5$ that uses $k = 3$ hash functions contains \( \text{"thisisavirus.com"} \)

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

<table>
<thead>
<tr>
<th>Index ( \to )</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>

\( h_1(\text{"thisisavirus.com"}) \to 2 \)
Bloom Filters: Example

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

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

contains(“thisisavirus.com”) = True

Index

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

function `contains(x)`

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

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

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

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

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

```
function CONTAINS(x)
    return t_1[h_1(x)] == 1 ∧ t_2[h_2(x)] == 1 ∧ ⋯ ∧ 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 $h_i(x)$, otherwise returns False.
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”)
Bloom Filters: False Positives

A 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”)

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

<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

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

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>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

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

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</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”)

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

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

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

Collision, is already set to 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>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</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** \( \text{CONTAINS}(x) \)

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

<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>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 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: Example

A bloom filter $t$ of length $m = 5$ that uses $k = 3$ hash functions 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>
Bloom Filters: Example

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

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

\[
\begin{align*}
\text{h}_1(\text{"verynormalsite.com"}) & \to 2 \\
\text{h}_2(\text{"verynormalsite.com"}) & \to 0 \\
\text{h}_3(\text{"verynormalsite.com"}) & \to 4 \\
\end{align*}
\]

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

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

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

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

\[
\begin{align*}
\text{Index} & \quad 0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 \\
\hline
\text{t}_1 & \quad 0 & \quad 1 & \quad 1 & \quad 0 & \quad 0 \\
\text{t}_2 & \quad 1 & \quad 1 & \quad 0 & \quad 0 & \quad 0 \\
\text{t}_3 & \quad 0 & \quad 0 & \quad 0 & \quad 0 & \quad 1
\end{align*}
\]

Since all conditions satisfied, returns True (incorrectly)
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.
## Comparison with Hash Tables - Space

- Google storing 5 million URLs, each URL 40 bytes.
- Bloom filter with \( k = 30 \) and \( m = 5,000,000 \)

### Hash Table

\[
5,000,000 \times 40 = 200,000,000 \text{ bytes}
\]

### Bloom Filter

\[
\frac{8 \times 10,000,000}{8} = 10,000,000 \text{ bytes}
\]

5% of space

False pos rate 0.06%
Comparison with Hash Tables - 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%

**Hash Table**

\[
102,000 \times 0.5 \text{ sec} = 51,000 \text{ sec}
\]

\[
\text{about 5%}
\]

**Bloom Filter**

\[
102,000 \times \frac{1}{1000} = 102 \text{ secs}
\]

\[
2000 \times 0.5 + 100,000 \times 0.03 \times 0.5
\]

\[
2402 \text{ sec}
\]
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 disk lookups
- Internet routers often use Bloom filters to track blocked IP addresses.
- And on and on...
Bloom Filters typical of….  

of randomized algorithms and randomized data structures.

- Simple
- Fast
- Efficient
- Elegant
- Useful!
Back to R.V.s....

LOTUS

Law Of The Unconscious Statistician
**Expectation of Random Variable**

**Definition.** Given a discrete RV $X: \Omega \rightarrow \mathbb{R}$, the *expectation* or *expected value* of $X$ is

$$E[X] = \sum_{\omega \in \Omega} X(\omega) \cdot Pr(\omega)$$

or equivalently

$$E[X] = \sum_{x \in \Omega_x} x \cdot Pr(X = x)$$

Intuition: “Weighted average” of the possible outcomes (weighted by probability)
Linearity of Expectation

**Theorem.** For any two random variables $X$ and $Y$

$$
\mathbb{E}(X + Y) = \mathbb{E}(X) + \mathbb{E}(Y).
$$

**Theorem.** For any random variables $X_1, \ldots, X_n$, and real numbers $a_1, \ldots, a_n, c \in \mathbb{R}$,

$$
\mathbb{E}(a_1X_1 + \cdots + a_nX_n + c) = a_1\mathbb{E}(X_1) + \cdots + a_n\mathbb{E}(X_n) + c.
$$
Computing complicated expectations

Often boils down to the following three steps

- **Decompose**: Finding the right way to decompose the random variable into sum of simple random variables
  \[ X = X_1 + \cdots + X_n \]
- **LOE**: Observe linearity of expectation.
  \[ \mathbb{E}(X) = \mathbb{E}(X_1) + \cdots + \mathbb{E}(X_n). \]
- **Conquer**: Compute the expectation of each \( X_i \)

Often, \( X_i \) are **indicator** (0/1) random variables.
Indicator random variable

For any event $A$, can define the indicator random variable $X$

$$X = \begin{cases} 
1 & \text{if event } A \text{ occurs} \\
0 & \text{if event } A \text{ does not occur}
\end{cases}$$

$$\mathbb{P}(X = 1) = \mathbb{P}(A)$$
$$\mathbb{P}(X = 0) = 1 - \mathbb{P}(A)$$

$$\mathbb{E}(X) = \mathbb{P}(A)$$
Example: Returning Homeworks

- Class with $n$ students, randomly hand back homeworks. All permutations equally likely.
- Let $X$ be the number of students who get their own HW.
- Let $Y = (X^2 + 4) \mod 8$.
- What is $\mathbb{E}(Y)$?

$$g(x) = (x^2 + 4) \mod 8$$

<table>
<thead>
<tr>
<th>$\Pr(\omega)$</th>
<th>$\omega$</th>
<th>$X(\omega)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/6</td>
<td>1, 2, 3</td>
<td>3</td>
</tr>
<tr>
<td>1/6</td>
<td>1, 3, 2</td>
<td>1</td>
</tr>
<tr>
<td>1/6</td>
<td>2, 1, 3</td>
<td>1</td>
</tr>
<tr>
<td>1/6</td>
<td>2, 3, 1</td>
<td>0</td>
</tr>
<tr>
<td>1/6</td>
<td>3, 1, 2</td>
<td>0</td>
</tr>
<tr>
<td>1/6</td>
<td>3, 2, 1</td>
<td>1</td>
</tr>
</tbody>
</table>

$$\mathbb{E}(Y) = \sum_{x \in \{0,1,3\}} g(x) \Pr(X=x)$$

$$= (3^2 + 4) \mod 8 \Pr(X=3) + (1^2 + 4) \mod 8 \Pr(X=1) + (0^2 + 4) \mod 8 \Pr(X=0)$$
Linearity is special!

In general $\mathbb{E}(g(X)) \neq g(\mathbb{E}(X))$

E.g., $X = \begin{cases} 1 \text{ with prob } 1/2 \\ -1 \text{ with prob } 1/2 \end{cases}$

- $\mathbb{E}(X^2) \neq \mathbb{E}(X)^2$

How DO we compute $\mathbb{E}(g(X))$?

\[ \mathbb{E}(X) = 0 \]
\[ 1 \cdot \frac{1}{2} + (-1) \cdot \frac{1}{2} \]
\[ \mathbb{E}(X^2) = 1 \]
Expectation of $g(X)$ (LOTUS)

**Definition.** Given a discrete RV $X: \Omega \rightarrow \mathbb{R}$, the expectation or expected value of $Y = g(X)$ is

$$E[Y] = \sum_{\omega \in \Omega} g(X(\omega)) \cdot \Pr(\omega)$$

or equivalently

$$E[Y] = \sum_{x \in \Omega_X} g(x) \cdot \Pr(X = x)$$

or equivalently

$$E[Y] = \sum_{y \in \Omega_Y} y \cdot \Pr(Y = y)$$