Overview

- Set membership data structures
- Why are false positives acceptable
- A Bloom filter in a few steps
- Bloom filter tricks
- GloBiMaps
SetMembership

Add \((x: U)\)
- **List**: \(O(1)\)
- **Tree**: \(O(\log n)\)
- **HashMap**: \(O(1)\), \(O(n)\)

Remove \((x: U)\)
- **List**: \(O(n)\)
- **Tree**: \(O(\log n)\)
- **HashMap**: \(O(1)\), \(O(n)\)

Test \((x: U) : bool\)
- **List**: \(O(n)\)
- **Tree**: \(O(\log n)\)
- **HashMap**: \(O(1)\), \(O(n)\)

"Search"
- Needs: \(= \quad \leq \quad \text{hash} = \)
Bit Set

- Bijection between elements and array of bits

Add (6) \( f \)
SetMembership

Add \((x: U)\)
- List: \(O(1)\)
- Tree: \(O(\log n)\)
- Hash Map: \(O(1)O(n)\)
- Bit Set: \(O(1)\)

Remove \((x, U)\)
- List: \(O(n)\)
- Tree: \(O(\log n)\)
- Hash Map: \(O(1)O(n)\)
- Bit Set: \(O(1)\)

Test \((x: U) : bool\)
- List: \(O(n)\)
- Tree: \(O(\log n)\)
- Hash Map: \(O(1)O(n)\)
- Bit Set: \(O(1)\)


gallow false positives

Needs: = \(\leq\) hash index

Space: \(O(n)\)

Also store the elements 1U1 bits
Space/Time Trade-offs in Hash Coding with Allowable Errors

Burton H. Bloom

Communications of the ACM • July 1970
In this paper trade-offs among certain computational factors in hash coding are analyzed. The paradigm problem considered is that of testing a series of messages one-by-one for membership in a given set of messages. Two new hash-coding methods are examined and compared with a particular conventional hash-coding method. The computational factors considered are the size of the hash area (space), the time required to identify a message as a nonmember of the given set (reject time), and an allowable error frequency.
In such applications, it is envisaged that overall performance could be improved by using a smaller core resident hash area in conjunction with the new methods and, when necessary, by using some secondary and perhaps time-consuming test to “catch” the small fraction of errors associated with the new methods. An example is discussed which illustrates possible areas of application for the new methods.
Allow false positives

Test(x) \rightarrow \text{NO} \rightarrow \text{FILTER} \rightarrow \text{NO} \rightarrow \text{True!}
Allow false positives

Test(x) → YES → FILTER

item ← true positive

→ YES → ACTUAL SET
Allow false positives

Test(x) → Filter → YES → Actual Set → NO

NO → YES

false positive
Is this a good idea?

- Depends on “Filter” versus “Actual Set” cost
- Depends on hit rates

- If filter can be amazingly small
  - Maybe you don’t actually have the set!
- Fast rejection most of the time
Applications 1/3

- Bloom ‘70: hyphenation
  - Most words covered by a few rules
  - Make a set containing the exceptions
  - Hyphenation algo: check set, else use rules
  - Let’s add a filter! False positives?
    - Unnecessary lookup; still correct
Applications 2/3

- **MrIlroy ‘82**: early UNIX spell-checkers
  - Store correct words. False positives?
    - Just accept them 😞
    - Amazingly small filter!
- **Spafford ‘92**: unsuitable passwords
  - Store the set. False positives?
    - Not really harmful
Applications 3/3

- **Chrome**: local filter for malicious URLs
  - Mostly misses
  - Google doesn’t see where you try to go
  - You don’t get the list
  - False positives?
    - Unnecessary warning; or ask Google.
Bit Set

- Good constant factors
- Too large when universe $U$ is large
  - Especially annoying if $n \ll |U|$
Add (x) \xrightarrow{h(x)} \text{m bits}

h : U \rightarrow \mathbb{Z}_m
Add \((y)\) \hline \(h(y)\) \hline

\[
h : U \rightarrow \mathbb{Z}_m
\]

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

\(m\) bits
$h : U \rightarrow \mathbb{Z}_m$

Test$(z)$ $\xrightarrow{h(z)}$

$z \in S$

Yes

No!

$m$ bits
$\text{Test}(x) \xrightarrow{h(x)} \text{m bits}$

$h : U \rightarrow \mathbb{Z}_m$

$x \in S$

$\text{Remove}(x)$?
The function $h$

- Take a “hash function”
- The function is deterministic
- For the analysis, we will assume it gives independent uniformly random indices
"Probabilistic data structure"

- Hashmaps: deterministic correctness, expected runtime

- Bloom filters: expected correctness, deterministic runtime
False positive probability

\[ P[ \text{bit } i \text{ is } 0 \text{ after the first insertion} ] \]

\[ 1 - \frac{1}{m} \]

\[ \lim_{m \to \infty} \left( 1 - \frac{1}{m} \right)^m = \frac{1}{e} \]

\[ P[ \text{bit } i \text{ is still } 0 \text{ after the first } n \text{ insertions} ] \]

\[ \left( 1 - \frac{1}{m} \right)^n = \left[ \left( 1 - \frac{1}{m} \right)^m \right]^{\frac{n}{m}} \approx e^{-\frac{n}{m}} \]
False positive probability

\[ P[ \text{bit } i \text{ is } 1 \text{ after } n \text{ insertions } ] \approx 1 - e^{-\frac{n}{3u}} \]

- Consider \( Test(x) \) for nonmember \( x \)
- Bit \( h(x) \) is set with probability...
Parameters

- $n$: Number of items
- $m$: Number of bits
- $\epsilon$: Error bound

- Probably fixed
- Maybe fixed
- Costs space
### Dictionary example

<table>
<thead>
<tr>
<th>English dictionary</th>
<th>( n )</th>
<th>( m )</th>
<th>( \varepsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(≈3 MB ASCII)</td>
<td>500,000</td>
<td>100 kB</td>
<td>≈ 46%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 MB</td>
<td>≈ 5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 MB</td>
<td>≈ 1.9%</td>
</tr>
</tbody>
</table>
$h_1, h_2: U \rightarrow \mathbb{Z}_m$

Add $(x) \xrightarrow{h_1(x)} h_2(x)$

$m$ bits

0 1 2 3 4 5 6 7

1 1 1 1
$h_1, h_2 : U \rightarrow \mathbb{Z}_m$

$z \in S$

\text{NO!}

\text{m bits}
Bloom filter

- Fix $k$ hash functions $h_i$
- Storage: array of $m$ bits, all start unset

$Add(x)$: set all bits $h_i(x)$
$Test(x)$: are all bits $h_i(x)$ set?
What is the effect of k?

- Increases runtime
- It does not affect the space!

- Error probability?
  - Check more bits: accidents less likely
  - Set more bits: accidents are more likely
False positive probability

- A particular bit is $0$ after the first insertion:
  $$(1 - \frac{1}{m})^k = \left[ (1 - \frac{1}{m})^m \right]^k \approx e^{-\frac{k}{m}}$$

- A particular bit is still $0$ after $n$ insertions:
  $$(1 - \frac{1}{m})^{kn} \approx e^{-\frac{kn}{m}}$$
False positive probability

- A particular bit is 1 after n insertions:
  \[ \approx 1 - e^{-\frac{kn}{m}} \]

- False positive Test \((x)\) (handwave!)
  \[ \left[ \frac{1 - (1 - \frac{1}{m})^{kn}}{k} \right]^k \approx (1 - e^{-\frac{kn}{m}})^k \]
Parameters

\( n \)  Number of items
\( m \)  Number of bits
\( k \)  Number of hash functions
\( \varepsilon \)  Error bound

- Probably fixed
- [Maybe fixed
- [Costs space
- Costs time
Picking $k$

- Idea: given $n$ and $m$, pick $k$ to minimize $\varepsilon$.

\[ \varepsilon \approx \left(1 - e^{-\frac{kn}{m}}\right)^k = \exp\left(k \ln\left(1 - e^{-\frac{kn}{m}}\right)\right) \]

\[ g \overset{\text{def}}{=} k \ln(1 - p) \]

\[ \frac{dg}{dk} \overset{!}{=} 0 \]
Picking $k$

- Idea: given $n$ and $m$, pick $k$ to minimize $\varepsilon$.

$$\varepsilon \approx \left(1 - e^{-\frac{kn}{m}}\right)^k = \exp\left(k \ln\left(1 - e^{-\frac{kn}{m}}\right)\right)$$

$$g \overset{\text{def}}{=} k \ln(1 - p) = -\frac{m}{n} \ln(p) \ln(1 - p)$$

$$= \frac{m}{n} \ln(p) \ln(1 - p)$$
Picking $k$

- Idea: given $n$ and $m$, pick $k$ to minimize $\varepsilon$. 

\[ \varepsilon \approx \left( 1 - e^{-\frac{kn}{m}} \right)^k = \exp\left( k \ln(1 - e^{-\frac{kn}{m}}) \right) \]

\[ \ln \Rightarrow g \overset{\text{def}}{=} k \ln(1 - p) = -\frac{m}{n} \ln(p) \ln(1-p) \]

\[ e^{-\frac{kn}{m}} = \frac{1}{2} \Rightarrow k = \ln 2 \cdot \frac{m}{n} \]

\[ \min \Rightarrow p = \frac{1}{2} \]
Picking $k$

- Idea: given $n$ and $m$, pick $k$ to minimize $\varepsilon$.

- Optimal: 
  \[ k = \left\lfloor \log_2 \frac{m}{n} \right\rfloor \]

\[ \varepsilon = \left(\frac{1}{2}\right)^k \approx 0.6185 \frac{m}{n} \]
Values of $k$

$m = 1 \text{ kb}$
Optimal $k$

$m = 1 \text{ kb}$
## Dictionary example

<table>
<thead>
<tr>
<th>English dictionary (∼3 MB ASCII)</th>
<th>n</th>
<th>m</th>
<th>k</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>500,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 kB</td>
<td>1</td>
<td>1</td>
<td></td>
<td>∼46 %</td>
</tr>
<tr>
<td>1 MB</td>
<td>1</td>
<td>1</td>
<td></td>
<td>∼5 %</td>
</tr>
<tr>
<td>3 MB</td>
<td>1</td>
<td>1</td>
<td></td>
<td>∼1.9 %</td>
</tr>
<tr>
<td>512 kB</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>∼11 %</td>
</tr>
<tr>
<td></td>
<td>6*</td>
<td></td>
<td>2</td>
<td>∼4 %</td>
</tr>
<tr>
<td></td>
<td>6*</td>
<td></td>
<td>6*</td>
<td>∼1 %</td>
</tr>
<tr>
<td>1 MB</td>
<td>12*</td>
<td></td>
<td>6*</td>
<td>∼0.03 %</td>
</tr>
<tr>
<td>3 MB</td>
<td>35*</td>
<td></td>
<td>12*</td>
<td>&lt;10(^{-10})</td>
</tr>
</tbody>
</table>