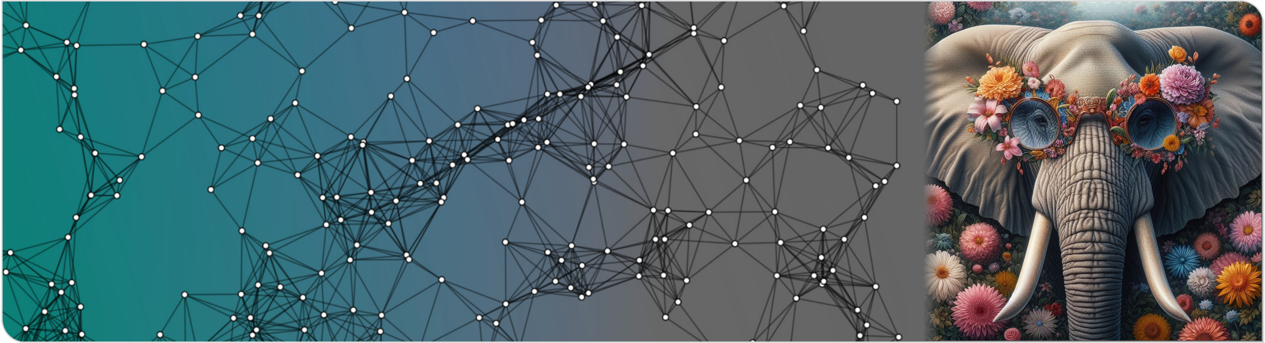
Probability and Computing – Bloom Filters

Stefan Walzer, Maximilian Katzmann | WS 2023/2024



Reminder: SUHA

Simple Uniform Hashing Assumption (SUHA)

- We have access to $h \sim \mathcal{U}(R^D)$ for any R and D.
- h takes $\mathcal{O}(1)$ time to evaluate.
- h takes no space to store.

Content

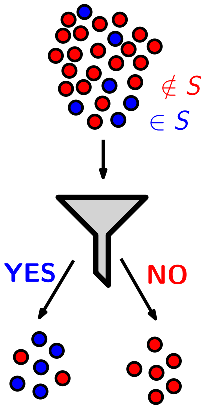
1. **What is a Filter or AMQ?**
   - Applications of Filters
2. **The Bloom Filter Data Structure**
3. **Analysis of Bloom Filters**
   - Expected fraction of zeroes in Bloom filters
   - Optimal tuning for Bloom filters
   - Main Theorem on Bloom filters

Filter = Approximate Membership Query Data Structure

Setting

- universe D of possible keys
- a set $S \subseteq D$ of $n = |S|$
- a false positive probability ε

Want: Data structure representing S.



Filter = Approximate Membership Query Data Structure

Setting

- universe D of possible keys
- a set $S \subseteq D$ of $n = |S|$
- a false positive probability ε

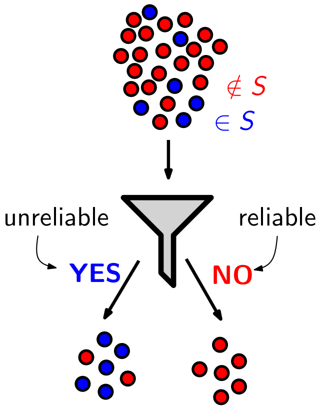
Want: Data structure representing S.

Operations

- **insert** elements to S and **delete** elements from S (optional)
- **query**: given $x \in D$ answer "is $x \in S$?" *approximately*:

$$\textbf{query}(x) = \textbf{YES} \text{ for } x \in S$$

$$\Pr[\textbf{query}(x) = \textbf{NO}] \geq 1 - \varepsilon \text{ for } x \notin S$$



Filter = Approximate Membership Query Data Structure

Karlsruhe Institute of Technology

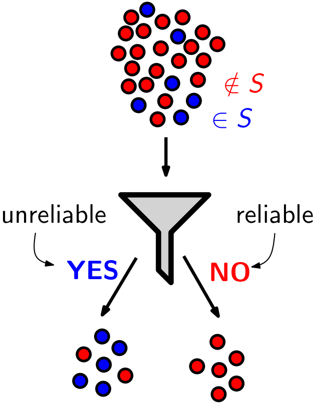
Setting

- universe D of possible keys
- a set $S \subseteq D$ of $n = |S|$
- a false positive probability ε

Want: Data structure representing S.

Space Requirement

- want $\mathcal{O}(n \log(1/\varepsilon))$ bits
- *much* smaller than $\mathcal{O}(n \log |D|)$ bits needed for hash table

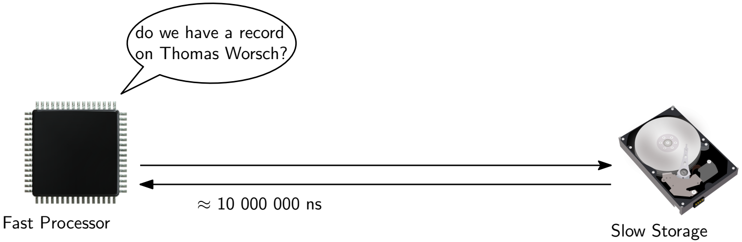


Operations

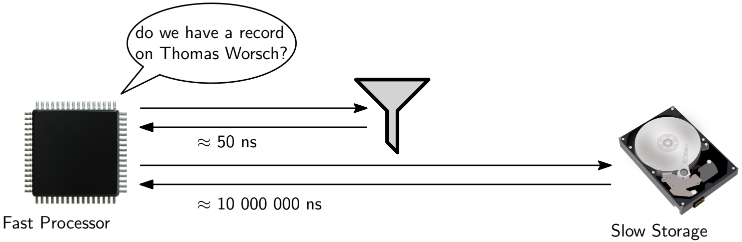
- **insert** elements to S and **delete** elements from S (optional)
- **query**: given $x \in D$ answer "is $x \in S$?" *approximately*:

$$\textbf{query}(x) = \textbf{YES} \text{ for } x \in S$$
$$\Pr[\textbf{query}(x) = \textbf{NO}] \geq 1 - \varepsilon \text{ for } x \notin S$$

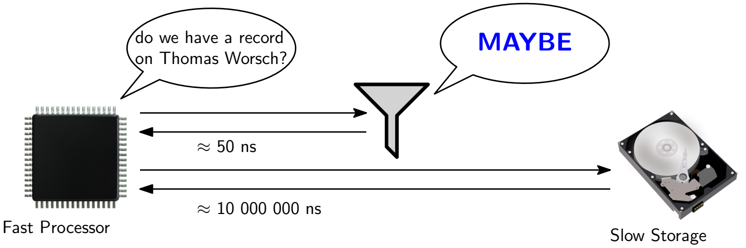
Applications of Filters



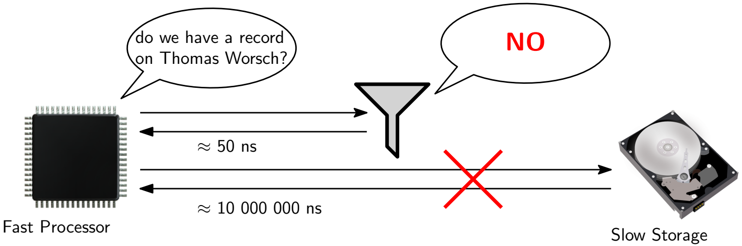
Applications of Filters



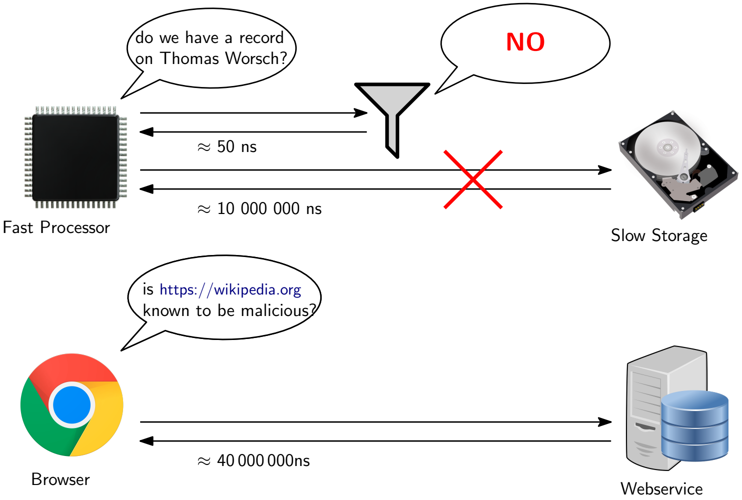
Applications of Filters



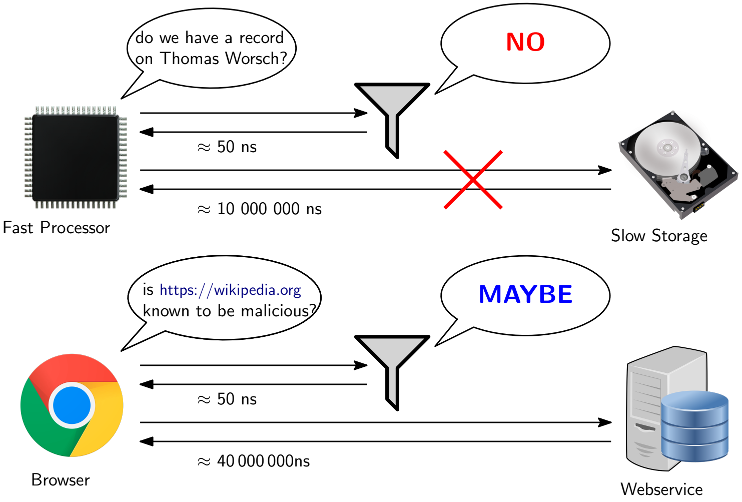
Applications of Filters



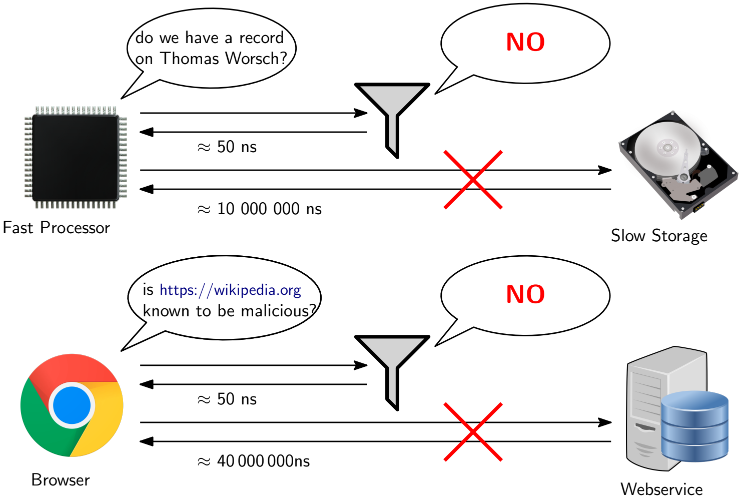
Applications of Filters



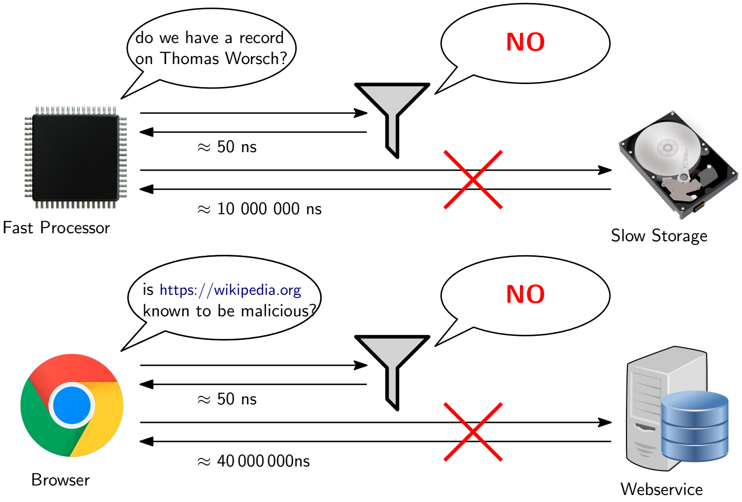
Applications of Filters



Applications of Filters



Applications of Filters



General Idea

If the reliable **NO** answers are frequent, a filter access can replace a (costly) access to a reliable data structure.

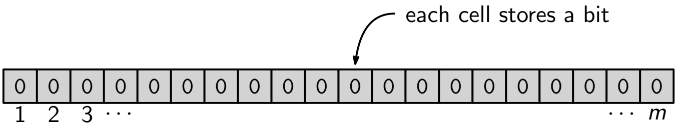
Content

The Bloom Filter Data Structure

Karlsruhe Institute of Technology

Parameters

m	length of a bit array $A[1..m]$ that we use
$k \in \mathcal{O}(1)$	number of hash functions $h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$
n	number of keys in $S \subseteq D$ (dynamic)
$\alpha \in \mathcal{O}(1)$	load n/m (dynamic)



insert(x):

for $i \in [k]$ **do**
  $A[h_i(x)] = 1$

query(x):

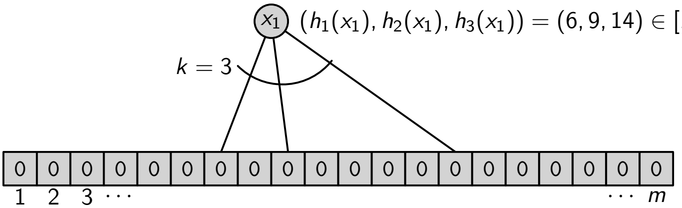
for $i \in [k]$ **do**
  if $A[h_i(x)] = 0$ **then**
    return NO
return YES

The Bloom Filter Data Structure

Parameters

m	length of a bit array $A[1..m]$ that we use
$k \in \mathcal{O}(1)$	number of hash functions $h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$
n	number of keys in $S \subseteq D$ (dynamic)
$\alpha \in \mathcal{O}(1)$	load n/m (dynamic)

$(h_1(x_1), h_2(x_1), h_3(x_1)) = (6, 9, 14) \in [m]^3$



insert(x):

for $i \in [k]$ **do**
  $A[h_i(x)] = 1$

query(x):

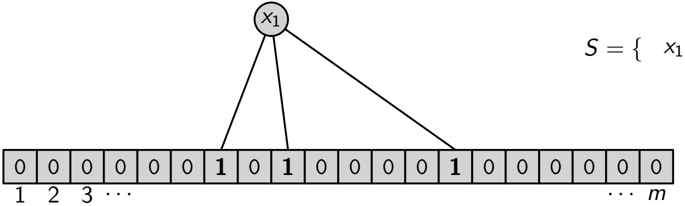
for $i \in [k]$ **do**
  if $A[h_i(x)] = 0$ **then**
    return NO

return YES

The Bloom Filter Data Structure

Parameters

m	length of a bit array $A[1..m]$ that we use
$k \in \mathcal{O}(1)$	number of hash functions $h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$
n	number of keys in $S \subseteq D$ (dynamic)
$\alpha \in \mathcal{O}(1)$	load n/m (dynamic)



$S = \{ \quad x_1, \qquad \}$

insert(x):

for $i \in [k]$ **do**
  $A[h_i(x)] = 1$

query(x):

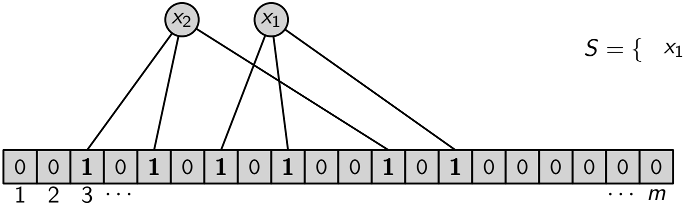
for $i \in [k]$ **do**
  if $A[h_i(x)] = 0$ **then**
    return NO
return YES

The Bloom Filter Data Structure

Karlsruhe Institute of Technology

Parameters

m	length of a bit array $A[1..m]$ that we use
$k \in \mathcal{O}(1)$	number of hash functions $h_1, \dots, h_k \sim \mathcal{U}([m]^D)$
n	number of keys in $S \subseteq D$ (dynamic)
$\alpha \in \mathcal{O}(1)$	load n/m (dynamic)



$S = \{ \quad x_1, x_2, \qquad \}$

insert(x):

for $i \in [k]$ **do**
  $A[h_i(x)] = 1$

query(x):

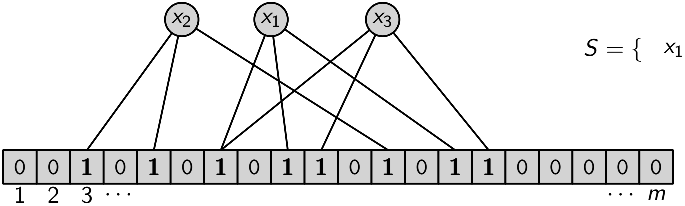
for $i \in [k]$ **do**
  if $A[h_i(x)] = 0$ **then**
    return NO
return YES

The Bloom Filter Data Structure

Karlsruhe Institute of Technology

Parameters

m	length of a bit array $A[1..m]$ that we use
$k \in \mathcal{O}(1)$	number of hash functions $h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$
n	number of keys in $S \subseteq D$ (dynamic)
$\alpha \in \mathcal{O}(1)$	load n/m (dynamic)



$S = \{ \; x_1, x_2, x_3, \quad \}$

insert(x):

for $i \in [k]$ **do**
  $A[h_i(x)] = 1$

query(x):

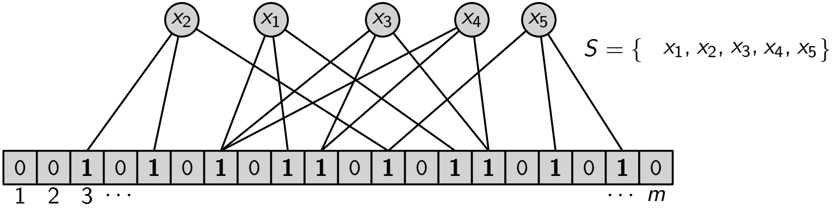
for $i \in [k]$ **do**
  if $A[h_i(x)] = 0$ **then**
    return NO
return YES

The Bloom Filter Data Structure

Karlsruhe Institute of Technology

Parameters

m	length of a bit array $A[1..m]$ that we use
$k \in \mathcal{O}(1)$	number of hash functions $h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$
n	number of keys in $S \subseteq D$ (dynamic)
$\alpha \in \mathcal{O}(1)$	load n/m (dynamic)



insert(x):

for $i \in [k]$ **do**
  $A[h_i(x)] = 1$

query(x):

for $i \in [k]$ **do**
  if $A[h_i(x)] = 0$ **then**
    return NO
return YES

The Bloom Filter Data Structure

Karlsruhe Institute of Technology

Parameters

m	length of a bit array $A[1..m]$ that we use
$k \in \mathcal{O}(1)$	number of hash functions $h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$
n	number of keys in $S \subseteq D$ (dynamic)
$\alpha \in \mathcal{O}(1)$	load n/m (dynamic)

0	0	**1**	0	**1**	0	**1**	0	**1**	**1**	0	**1**	0	**1**	**1**	0	**1**	0	**1**	0
1	2	3	$\cdots$															$\cdots$	m

insert(x):

for $i \in [k]$ **do**
  $A[h_i(x)] = 1$

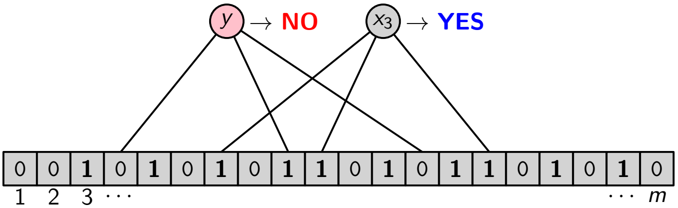
query(x):

for $i \in [k]$ **do**
  if $A[h_i(x)] = 0$ **then**
    return NO
return YES

The Bloom Filter Data Structure

Parameters

m	length of a bit array $A[1..m]$ that we use
$k \in \mathcal{O}(1)$	number of hash functions $h_1, \ldots, h_k \sim \mathcal{U}([m]^D)$
n	number of keys in $S \subseteq D$ (dynamic)
$\alpha \in \mathcal{O}(1)$	load n/m (dynamic)



insert(x):

for $i \in [k]$ **do**
  $A[h_i(x)] = 1$

query(x):

for $i \in [k]$ **do**
  if $A[h_i(x)] = 0$ **then**
    return NO
return YES

Content

Preparation

Exercise: Some approximations of e

$$\forall n \in \mathbb{N} : (1 + \tfrac{1}{n})^n \leq e \leq (1 + \tfrac{1}{n})^{n+1}$$
$$\text{and} \quad (1 - \tfrac{1}{n})^n \leq e^{-1} \leq (1 - \tfrac{1}{n})^{n-1}.$$

Corollaries

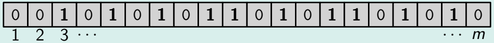
$$\forall n \in \mathbb{N} : (1 + \tfrac{1}{n})^n = e - \mathcal{O}(1/n)$$
$$\text{and} \quad (1 - \tfrac{1}{n})^n = e^{-1} - \mathcal{O}(1/n).$$

Bloom Filter Analysis (i)

Lemma

Assume $S = \{x_1, \dots, x_n\}$ is inserted into the Bloom filter. Let $(A_1, \dots, A_m) \in \{0,1\}^m$ be the random filter state and $Z := \sum_{i=1}^{m}(1 - A_i)$ the number of zeroes. Then

- i $\mathbb{E}[\frac{Z}{m}] = (1 - \frac{1}{m})^{m\alpha k} = e^{-\alpha k} - o(1)$
- ii For $y \notin S$: $\Pr[\textbf{query}(y) = \textbf{YES} \mid Z = z] = (1 - \frac{z}{m})^k$



Bloom Filter Analysis (i)

Lemma

Assume $S = \{x_1, \ldots, x_n\}$ is inserted into the Bloom filter. Let $(A_1, \ldots, A_m) \in \{0,1\}^m$ be the random filter state and $Z := \sum_{i=1}^{m}(1 - A_i)$ the number of zeroes. Then

- i $\mathbb{E}[\frac{Z}{m}] = (1 - \frac{1}{m})^{m\alpha k} = e^{-\alpha k} - o(1)$
- ii For $y \notin S$: $\Pr[\textbf{query}(y) = \textbf{YES} \mid Z = z] = (1 - \frac{z}{m})^k$

0	0	**1**	0	**1**	0	**1**	0	**1**	**1**	0	**1**	0	**1**	**1**	0	**1**	0	**1**	0
1	2	3 ···																···	m

Proof of (i).

$$\mathbb{E}[\tfrac{Z}{m}] = \frac{1}{m}\mathbb{E}[\sum_{i=1}^{m}(1 - A_i)] = \frac{1}{m}\sum_{i=1}^{m}\Pr[A_i = 0] = \frac{1}{m}\sum_{i=1}^{m}\Pr[A_1 = 0] = \Pr[A_1 = 0]$$

$$= \Pr[\forall x \in S : \forall i \in [k] : h_i(x) \neq 1] \overset{SUHA}{=} \prod_{x \in S}\prod_{i \in [k]}\Pr[h_i(x) \neq 1] \overset{SUHA}{=} \prod_{x \in S}\prod_{i \in [k]}(1 - \tfrac{1}{m})$$

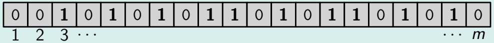
$$= (1 - \tfrac{1}{m})^{nk} = (1 - \tfrac{1}{m})^{m\alpha k} = (e^{-1} - o(1))^{\alpha k} = e^{-\alpha k} - o(1).$$

Bloom Filter Analysis (i)

Lemma

Assume $S = \{x_1, \dots, x_n\}$ is inserted into the Bloom filter. Let $(A_1, \dots, A_m) \in \{0,1\}^m$ be the random filter state and $Z := \sum_{i=1}^{m}(1 - A_i)$ the number of zeroes. Then

- i $\mathbb{E}[\frac{Z}{m}] = (1 - \frac{1}{m})^{m\alpha k} = e^{-\alpha k} - o(1)$
- ii For $y \notin S$: $\Pr[\textbf{query}(y) = \textbf{YES} \mid Z = z] = (1 - \frac{z}{m})^k$



Proof of (ii).

$$\Pr[\textbf{query}(y) = \textbf{YES} \mid Z = z] = \Pr[\forall i \in [k] : A_{h_i(y)} = 1 \mid Z = z] = \prod_{i \in [k]} \left(\frac{m - z}{m}\right) = (1 - \tfrac{z}{m})^k.$$

How should a Bloom filter be configured?

Approximate false positive rate

From the previous Lemma we get for $y \notin S$:

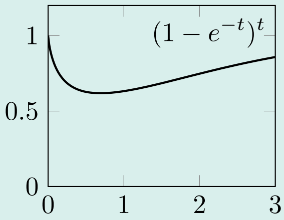
$$\varepsilon = \Pr[\textbf{query}(y) = \textbf{YES}] \approx \Pr[\textbf{query}(y) = \textbf{YES} \mid Z = \mathbb{E}[Z]]$$
$$\overset{\text{ii}}{=} \left(1 - \frac{\mathbb{E}[Z]}{m}\right)^k \overset{\text{i}}{=} (1 - e^{-\alpha k} + o(1))^k \approx (1 - e^{-\alpha k})^k.$$

Which k minimises ε? (when α is fixed)

$$\begin{aligned} &\arg\min_{k \in \mathbb{N}} (1 - e^{-\alpha k})^k \\ =\ &\arg\min_{k \in \mathbb{N}} (1 - e^{-\alpha k})^{\alpha k} \\ \approx\ &\frac{1}{\alpha} \arg\min_{t \in \mathbb{R}_+} (1 - e^{-t})^t \\ =\ &\frac{1}{\alpha} \arg\min_{t \in \mathbb{R}_+} t \ln(1 - e^{-t}) \end{aligned}$$

- plot $(1 - e^{-t})^t \rightsquigarrow$ one global minimum.
- deriving $t \ln(1 - e^{-t})$ gives $\ln(1 - e^{-t}) + \frac{te^{-t}}{1 - e^{-t}}$
- $t = \ln(2)$ is root of the derivative.

$\hookrightarrow$ $k = \ln(2)/\alpha$ is optimal for fixed α.
$\hookrightarrow$ choose α and k such that $\alpha k = \ln(2)$



False Positive Probability and Space

Karlsruhe Institute of Technology

Intuition for optimality of $\alpha k = \ln(2)$

- gives $\mathbb{E}[\frac{Z}{m}] \approx e^{-\alpha k} = \frac{1}{2}$
- maximises *entropy* of the filter bits

Theorem

A Bloom filter with $k \in \mathbb{N}$ hash functions and load factor $\alpha = \ln(2)/k$ has

space requirement $m = n/\alpha = \frac{kn}{\ln 2} \approx 1.44kn$ bits and

false positive probability $\varepsilon = 2^{-k} + o(1)$.

- space requirement ✓
- false positive probability: need a concentration bound first.

Concentration bound for Z

Lemma

- i $\Pr[Z \leq \mathbb{E}[Z] - \delta \quad] \leq \exp(-\Theta(\delta^2/m))$ for any $\delta > 0$,
- ii $\Pr[Z \leq \mathbb{E}[Z] - m^{2/3}] \leq \exp(-\Theta(m^{1/3}))$ by setting $\delta = m^{2/3}$.

Concentration bound for Z

Reminder: McDiarmid's Inequality

If $X_1, \dots, X_n$ are indepedent and f satisfies the bounded difference property with parameters $(\Delta_i)_{i\in[n]}$ then

$$\Pr[\mathbb{E}[f(X_1,\dots,X_n)] - f(X_1,\dots,X_n) \geq \delta] \leq \exp\left(\frac{-2\delta^2}{\sum_{i=1}^n \Delta_i^2}\right).$$

Lemma

- i $\Pr[Z \leq \mathbb{E}[Z] - \delta \quad] \leq \exp(-\Theta(\delta^2/m))$ for any $\delta > 0$,
- ii $\Pr[Z \leq \mathbb{E}[Z] - m^{2/3}] \leq \exp(-\Theta(m^{1/3}))$ by setting $\delta = m^{2/3}$.

Proof of (i) using the method of bounded differences.

- Z is a function of kn independent hash values
- each hash value can change Z by at most 1
- use method of bounded differences!

$$\Rightarrow \Pr[Z \leq \mathbb{E}[Z] - \delta] \leq \Pr[\mathbb{E}[Z] - Z \geq \delta] = \exp\left(\frac{-2\delta^2}{nk}\right) = \exp\left(\frac{-2\delta^2}{m\alpha k}\right) = \exp\left(\frac{-2\delta^2}{m\ln(2)}\right). \quad \square$$

False Positive Probability of Bloom filters

Karlsruhe Institute of Technology

Proof of the Main Theorem on Bloom filters (false positive probability).

By choice of k and α we have $\mathbb{E}[\frac{Z}{m}] = e^{-\alpha k} - o(1) = \frac{1}{2} - o(1)$.
Let $y \notin S$ and $B = \lfloor \mathbb{E}[Z] - m^{2/3} \rfloor$.

$$\Pr[\textbf{query}(y) = \textbf{YES}] \stackrel{\text{LTP}}{=} \sum_{z=1}^{m} \Pr[Z = z] \cdot \Pr[\textbf{query}(y) = \textbf{YES} \mid Z = z] = \sum_{z=1}^{m} \Pr[Z = z] \cdot (1 - \tfrac{z}{m})^k$$

$$\leq \sum_{z=1}^{B} \Pr[Z = z] + \sum_{z=B+1}^{m} \Pr[Z = z](1 - \tfrac{B+1}{m})^k \leq \Pr[Z \leq B] + (1 - \tfrac{B+1}{m})^k$$

$$\leq \Pr[Z \leq \mathbb{E}[Z] - m^{2/3}] + \left(1 - \frac{\mathbb{E}[Z] - m^{2/3}}{m}\right)^k \stackrel{\text{ii}}{\leq} \exp(-\Theta(m^{1/3})) + (1 - \tfrac{1}{2} + o(1))^k = 2^{-k} + o(1). \quad \square$$

How to Configure Your Bloom Filter

Theorem

A Bloom filter with $k \in \mathbb{N}$ hash functions and load factor $\alpha = \ln(2)/k$ has

space requirement $m = n/\alpha = \frac{kn}{\ln 2} \approx 1.44kn$ bits and

false positive probability $\varepsilon = 2^{-k} + o(1)$.

How to determine m and k (the parameters you actually need)

1. n: determined by input
2. ε: choose a trade-off between space usage and false positive probability
   - If utility comes from negative answers "$x \notin S$, definitely" and running time is negligible, then:
     - want to maximise utility − disutility, where: ($\propto$ means "proportional to")
     - utility $\propto \frac{\text{negative answers}}{\text{second}} = \frac{\text{queries}}{\text{second}} \cdot \Pr[x \notin S] \cdot (1 - \varepsilon)$
     - disutility $\propto$ space consumption $= 1.44 \log(1/\varepsilon) n$ bits of RAM or cache
3. compute $k = \lceil \log(1/\varepsilon) \rceil$ // effectively restricts ε to powers of 2
4. compute $\alpha = \ln(2)/k$ and $m = \lceil n/\alpha \rceil$

Remarks

Much, much more is known

- more functionality
  - $\hookrightarrow$ counting Bloom filters support deletions
- better space efficiency
  - $\hookrightarrow$ cuckoo filters use $n \log(1/\varepsilon) + \mathcal{O}(n)$ bits rather than $\approx 1.44 n \log(1/\varepsilon)$ bits
  - $\hookrightarrow$ static filters (no insertions or deletions) use $n \log(1/\varepsilon) + o(n)$ bits.
- better query times
  - $\hookrightarrow$ blocked Bloom filters improve cache efficiency
- ...

Conclusion

Karlsruhe Institute of Technology

- **Approximate Membership Queries.**
  - Decide “is $x \in S$?” with *false positive probability* ε.
  - The Bloom filter is the most widespread AMQ.
- **Space Efficient.** $\approx 1.44 \log(1/\varepsilon)$ bits per element
  - often fit into cache or RAM when proper set data structure does not
- **Used to prevent costly accesses.**
  - Reliable on **NO** answers.
  - Useful if **NO** answers are frequent.

Anhang: Mögliche Prüfungsfragen I

- Approximate-Membership-Query Datenstrukturen im Allgemeinen
  - Welche Aufgabe hat eine AMQ Datenstruktur?
  - Was ist der Vorteil gegenüber einer exakten Datenstruktur?
  - Was wäre ein Anwendungsfall, in dem eine AMQ Datenstruktur nützlich ist?
- Bloomfilter
  - Wie ist ein Bloomfilter aufgebaut und welche Operationen unterstützt er?
  - Welche Parameter gibt es, und wie hängen diese zusammen?
  - Was hat unsere Analyse zur geschickten Wahl der Parameter zu sagen? Wie werden die übrigen Parameter gewählt? Welcher Speicherverbrauch ergibt sich?
  - Fragen zur Analyse
    - Welche Anzahl von Nullen bzw. Einsen erwarten wir?
    - Wie hängt die falsch-positiv Wahrscheinlichkeit mit der Anzahl Nullen bzw. Einsen zusammen?
    - Wir kann man argumentieren, dass die Anzahl Nullen bzw. Einsen im Bloomfilter nahe am Erwartungswert liegt?

Ablauf der letzten Termine

Inverted Classroom Grundidee

- Zu Hause: Videovorlesung gucken.
- Vor Ort: Übungsaufgaben mit Hilfestellung bearbeiten.
  $\hookrightarrow$ Weniger oder keine Übungen mehr zuhause.

Ablauf der restlichen Termine

- **Di 30.1: reguläre Übung zu Blättern 10 + 11** (mit Hans-Peter)
- Video gucken (Cuckoo Hashing, 30 min)
- Blatt 12 abgeben (Bloom Filter, nur 1 Aufgabe)
- **Do 1.2: reguläre Übung zu Blatt 12, Bearbeitung von Blatt 13 zu Cuckoo Hashing** (mit Stefan)
- Video gucken (Peeling)
- Blatt 13 (finalisieren und) abgeben
- **Do 8.2: Bearbeitung von Blatt 14 zu Peeling** (mit Stefan)
- Video gucken (Perfect Hashing)
- Blatt 14 (finalisieren und) abgeben
- **Di 13.2: Bearbeitung von Blatt 15 zu Perfect Hashing** (mit Stefan)
- Blatt 15 (finalisieren und) abgeben
- **Do 15.2: Termin reserviert für Fragen, Prüfungsmodalitäten usw.** (mit allen)