Bloom Filters

A Problem

Lookup five-word sequences, return count (or not found)

Most are misses (not found)

A Problem

Lookup five-word sequences, return count (or not found)

Most are misses (not found)

In General

Distributed storage

New keys are broadcast (or read-only)

High miss rate

A Problem

Lookup five-word sequences, return count (or not found)

Most are misses (not found)

In General

Distributed storage

New keys are broadcast (or read-only)

High miss rate

Handle some misses locally

$$\Longrightarrow$$

Reduce network

Bloom Filter

Represent a set, probabilistically.

`insert(key)` Add `key` to the set.

`query(key)` If `key` is in the set, return `maybe`.
If `key` is not in the set, return `no` or `maybe`.

Bloom Filter

Represent a set, probabilistically.

`insert(key)` Add `key` to the set.

`query(key)` If `key` is in the set, return `maybe`.
If `key` is not in the set, return `no` or `maybe`.

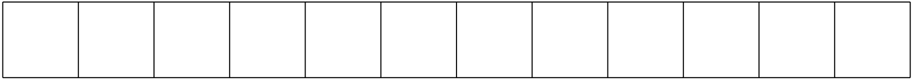
Usage

Ask a Bloom filter locally.

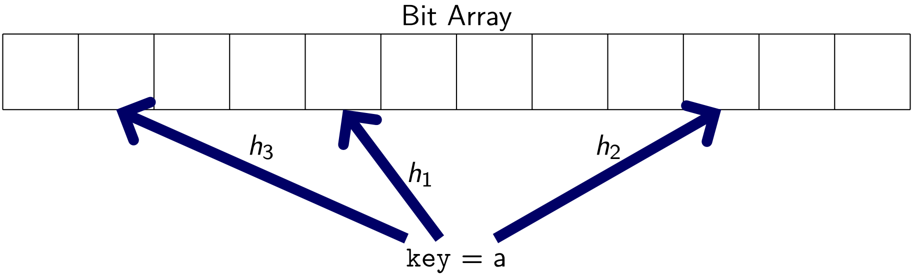
`no` Key is definitely not found $\rightarrow$ avoid network.

`maybe` Ask the network.

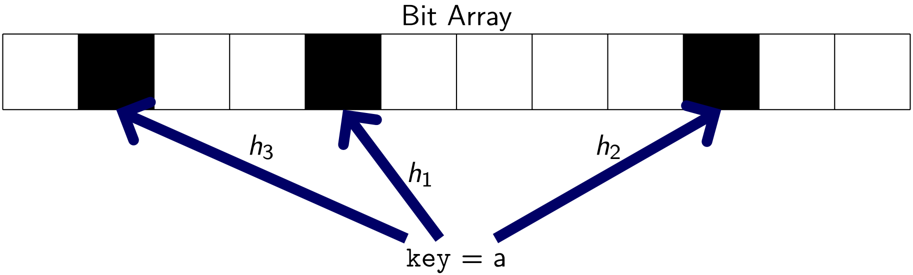
Bit Array



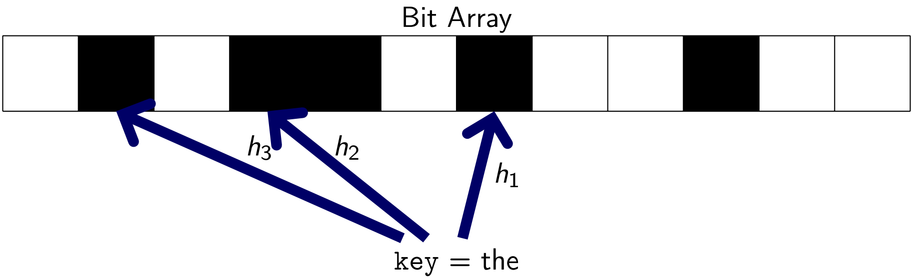
1. Initially the array is all 0s.



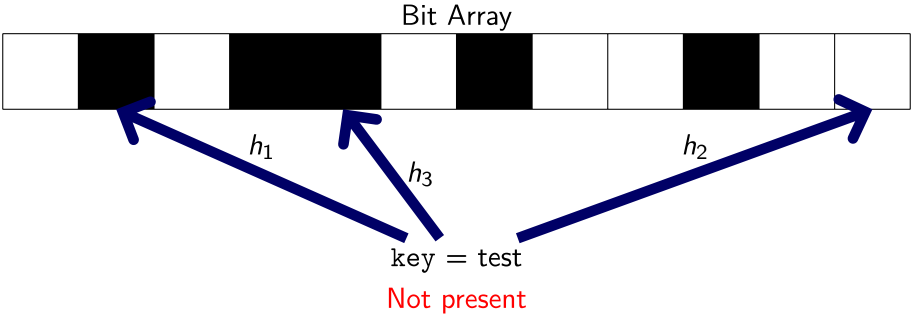
1. Initially the array is all 0s.
2. Hash functions assign bit positions to keys.



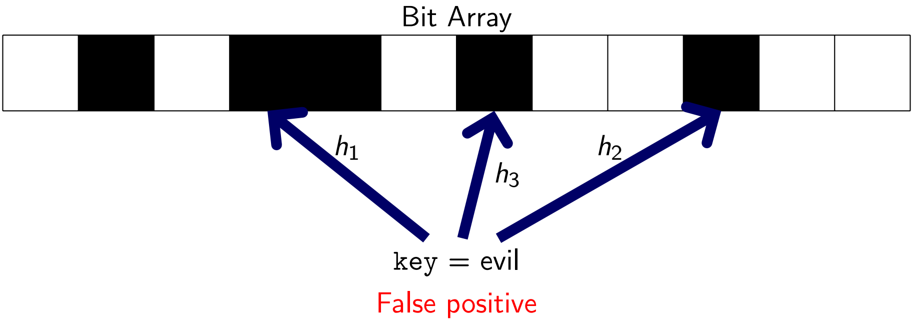
1. Initially the array is all 0s.
2. Hash functions assign bit positions to keys.
3. Insertion sets the corresponding bits to 1.



1. Initially the array is all 0s.
2. Hash functions assign bit positions to keys.
3. Insertion sets the corresponding bits to 1.



1. Initially the array is all 0s.
2. Hash functions assign bit positions to keys.
3. Insertion sets the corresponding bits to 1.
4. Queries check that the corresponding bits are 1.



1. Initially the array is all 0s.
2. Hash functions assign bit positions to keys.
3. Insertion sets the corresponding bits to 1.
4. Queries check that the corresponding bits are 1.

Bloom Filters: memory efficient
. . . but some probability of false positives.

Bloom Filters: memory efficient
. . . but some probability of false positives.

Not done yet:

- Need multiple hash functions.
- What is the false-positive probability?
- How many hash functions?

Multiple Hash Functions?

We need independent hash functions:

$$h_1(\texttt{the}), h_2(\texttt{the}), h_3(\texttt{the}), \ldots$$

Multiple Hash Functions?

We need independent hash functions:

$$h_1(\texttt{the}), h_2(\texttt{the}), h_3(\texttt{the}), \ldots$$

Just use one good hash function h and concatenate with key:

$$h(\texttt{1\_key}), h(\texttt{2\_key}), h(\texttt{3\_key}), \ldots$$

The optimal number of hashes is

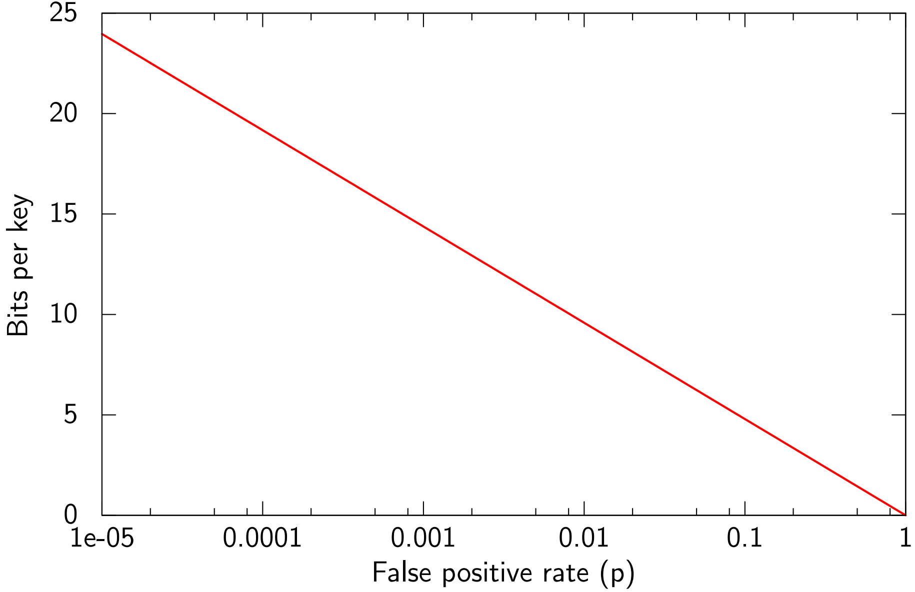
$$\text{hashes} \approx \frac{\text{bits}}{\text{entries}} \ln 2$$

To satisfy false-positive probability p, Bloom filters use

$$\approx \frac{-\log_2 p}{\ln 2}$$

bits per key.

Don't worry about the exact equations.
But deriving them is fun!


25
20
15
10
5
0
Bits per key
1e-05
0.0001
0.001
0.01
0.1
1
False positive rate (p)

Summary

Approximately represent a very large set in small memory.

Used to reduce expensive lookups in SSTable, BigTable, ...

Also useful in isolation for error-tolerant tasks.