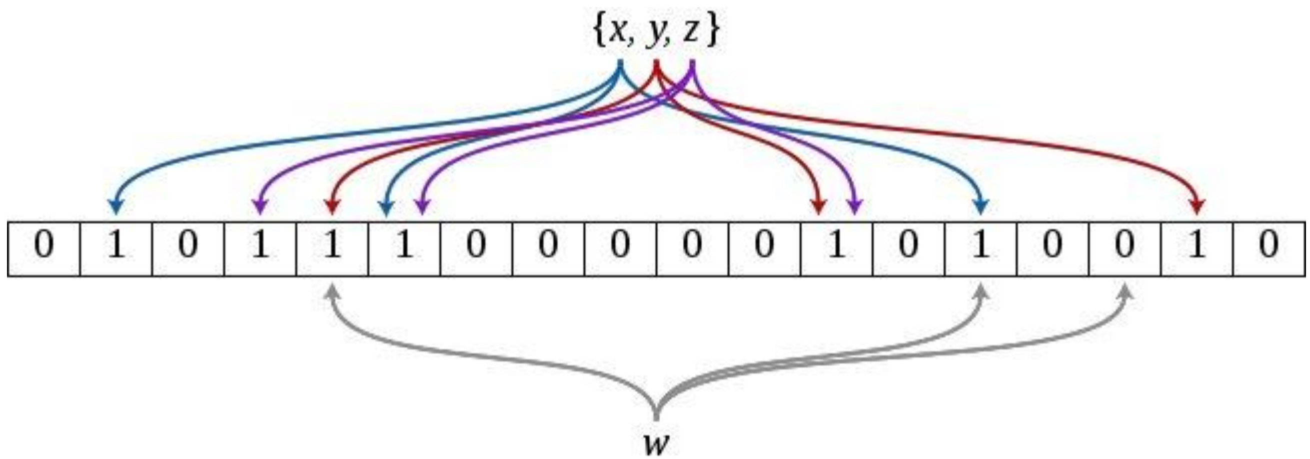
CS2040S – Data Structures and Algorithms

Lecture 8* – Bloom Filter

chongket@comp.nus.edu.sg

National University of Singapore
School of Computing

Bloom Filter – A Probabilistic Map



Motivation for using Bloom Filter

- When a hash table (specifically a **hash set**) is too large to fit into RAM, data will have to be read/written to and from hard disk which is much slower than RAM

- A possible solution is to use Bloom Filter instead of a classical hash table, as it is able to “compress” the amount of memory required to store the data

Introduction to Bloom Filter (1)

- Most important characteristic of a Bloom Filter is that retrieval of a key in the Bloom Filter is probabilistic
  - “No” means the key is definitely not in the filter
  - “Yes” has a certain probability that the key is actually not in the Bloom Filter (a false positive)

Introduction to Bloom Filter (2)

- Data Structure used – A bit array (in Java there is no bit primitive data type so have to use the **BitSet** class) which is initialized to 0
- Basic Operations
  – Insertion
  – Retrieval
- Can’t do removal in a Bloom Filter !

Bloom Filter - Insert operation (1)

- Inserting an integer key k into the bloom filter requires setting up to M bits of the bit array to 1. To do this,
  - use M independent and uniformly distributed hash functions $H_1(k)$ to $H_M(k)$
  - Each hash function returns an (hopefully different) index of the bit array which is then set to 1

- M is usually bounded by a small value so can be considered a constant

Bloom Filter - Insert operation (2)

3 hash functions:
$H_1(key) = (key\%17)\%10$
$H_2(key) = (key\%29)\%10$
$H_3(key) = (key\%37)\%10$

0	0	0	0	0	0	0	0	0	0

Bit array of size 10

Inserting 123:
$H_1(123) = 4$
$H_2(123) = 7$
$H_3(123) = 2$

{123}

0	0	1	0	1	0	0	1	0	0

Bit array of size 10

Inserting 306:
$H_1(306) = 0$
$H_2(306) = 6$
$H_3(306) = 0$

{123,306}

1	0	1	0	1	0	1	1	0	0

Bit array of size 10

Bloom Filter - Insert operation (3)

Inserting 7125:
$H_1(7125) = 2$
$H_2(7125) = 0$
$H_3(7125) = 1$

{123,306,7125}

1	1	1	0	1	0	1	1	0	0

Bit array of size 10

Overlap with bit representation for 123, 306 with 7125

Bloom Filter - Retrieval operation (1)

- To retrieve a key again use the M hash functions to get M indices in the filter
  - If all M indices are set to 1 then the key is possibly in the filter
  - If at least one index is 0 then the key is definitely not in the filter

Bloom Filter - Retrieval operation (2)

{123,306,7125}

Given

1	1	1	0	1	0	1	1	0	0

Bit array of size 10

Retrieving 7125:
$H_1(7125) = 2$
$H_2(7125) = 0$
$H_3(7125) = 1$

{123,306,7125}

1	1	1	0	1	0	1	1	0	0

Bit array of size 10

All 1's so return 7125 is in the Bloom Filter

Bloom Filter - Retrieval operation (3)

Retrieving 513:

$H_1(513) = 3$

$H_2(513) = 0$

$H_3(513) = 2$

{123,306,7125}

1	1	1	0	1	0	1	1	0	0

Bit array of size 10

Retrieving 74:

$H_1(74) = 6$

$H_2(74) = 6$

$H_3(74) = 0$

{123,306,7125}

1	1	1	0	1	0	1	1	0	0

Bit array of size 10

Considerations when designing a good Bloom Filter (1)

- In general, the bigger the bloom filter and the more hash functions used, the less likely the bits set for any key will coincide with any other key
- However this defeats the purpose of the bloom filter to “compress” the memory requirements
- Need to find a good compromise

Considerations when designing a good Bloom Filter (2)

- 4 variables for a bloom filter
  - N: the number of keys to be inserted
  - S: the size of the bit array
  - M: the number of hash functions used
  - P: the probability of a false positive

- If we fix the value for N or upper bound for N, and the required P, then S and M can be computed as follows

Considerations when designing a good Bloom Filter (3)

$$S = Ceil\left(-\frac{N * \ln P}{(\ln 2)^2}\right)$$

$$M = round\left(\frac{S}{N} * \ln 2\right)$$

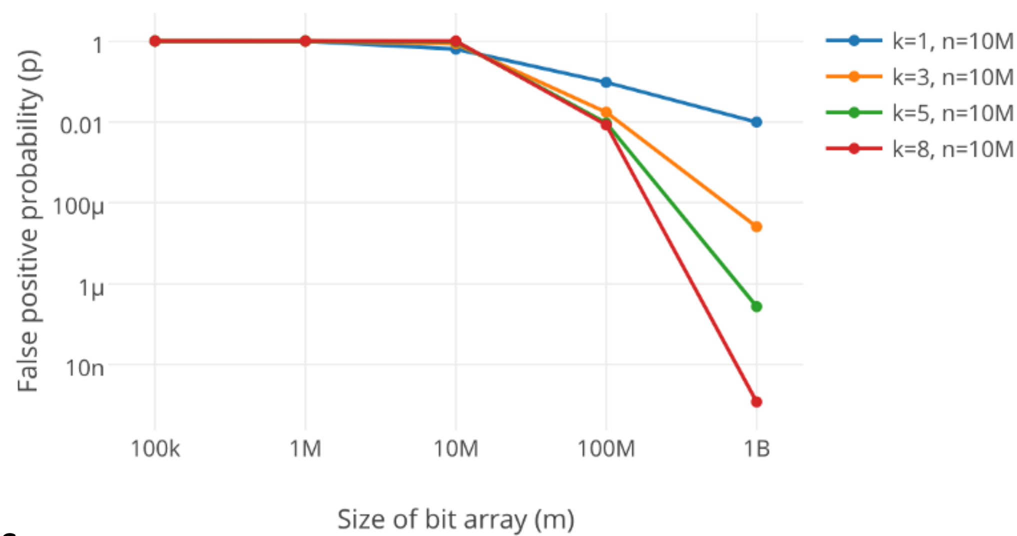


Image Credit: [Abishek Bhat's article about bloom filter]

Compression Ratio :

Assuming keys = 32 bits or 4 byte integers
At N = 10M and P = 0.01
Bit array size = 100Mbit = 12.5 Mbytes

If use hashtable need at least 40Mbytes.

Compression ratio = 40/12.5 = **3.2 times**

Considerations when designing a good Bloom Filter (4)

- In general the number of bytes required to store each key in a bloom filter with false positive rate P is

$$Average\ \#\ of\ bytes\ per\ key = \frac{S}{N} = 0.26 \ln \frac{1}{P}$$

- The number of bytes required per key is independent of the size of the key itself, thus the greater the size of a key the greater the compression ratio

Bloom Filter Conclusion

- Pros
  - Useful when number of keys too large for classical hash table to fit into RAM
  - Useful when we can accept a small amount of errors (false positives) during retrieval
  - Use M bits to represent a key regardless of the size of the key (especially useful when key is a long string)
- Cons
  - Larger overhead to insert/retrieve a key since need to compute M hash functions instead of 1 hash function
  - Not useful for problems where no error is tolerated
  - Not cache-friendly since the M bits for a key may not be close together thus cannot fit into cache