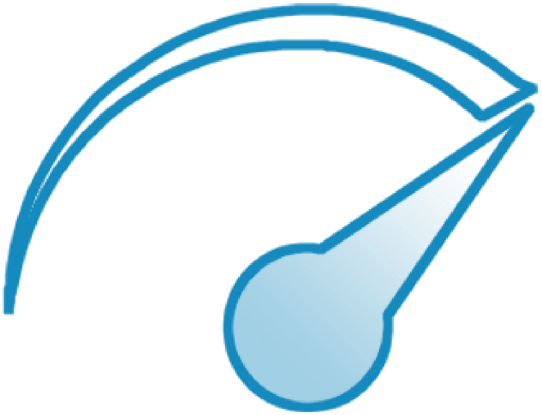
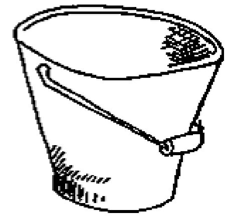
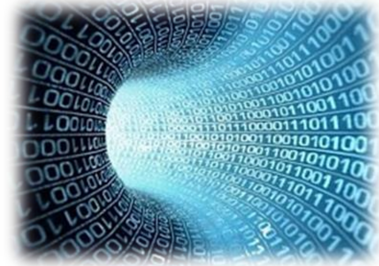
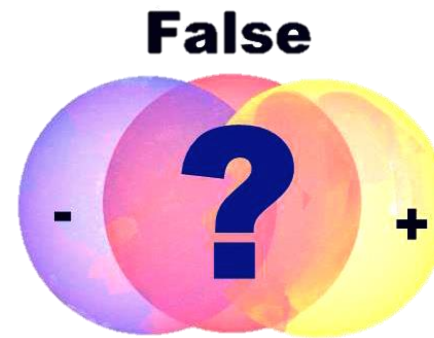






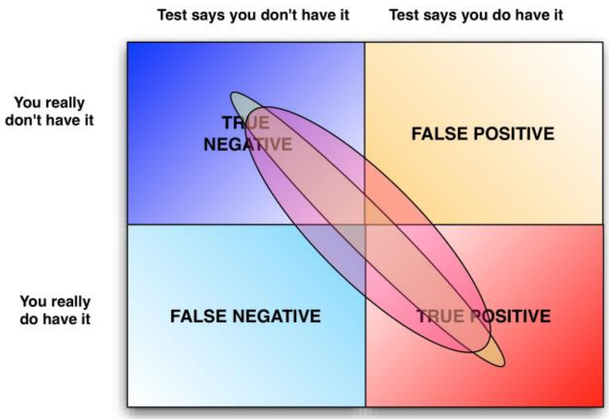


BLOOM Filter
Lightweight Data Structure

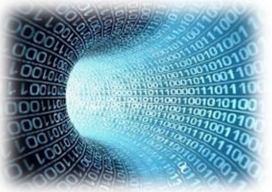
Traduction en cours

Sketching Data Structures

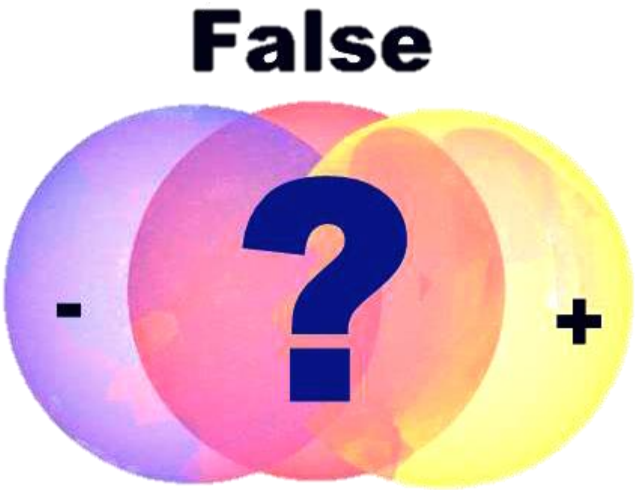
- The main feature of sketching data structures is that they can answer certain questions about the data extremely efficiently, at the price of the occasional error.



False Positif

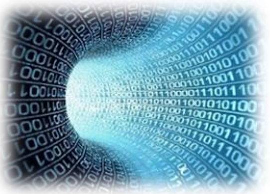


- Bloom filter allows to get away with much less storage at the price of an occasional errors.
- Errors can only be a false positive
- Bloom Filter might say that x is in A when in fact it is not.
- When BF says that x is not in A,
  - This is always true,
  - False negatives are impossible.



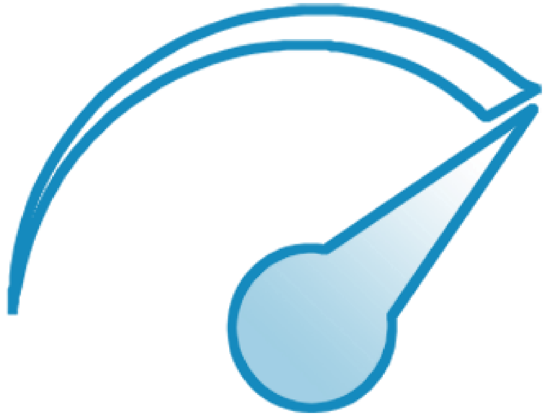
Bloom filters

- Bloom Filter is representing which elements are present in a set.
- Bloom filters are used to reduce expensive disk (or network) lookups for non-existent keys.
- The Bloom Filter is a data structure that compactly represents a set as a bitmap which is updated via hashing.



Bloom Filter common usage

- Browser tests for malicious website
- Test for weak passwords
- Propose a list of items not seen by a given user
- Bitcoin
- Web cache hit cache



Bloom Filter sub-linear space

- Bloom filter doesn’t store the actual elements, it stores the “membership” of them.
- It uses sub-linear space opening the possibility for false positives, meaning there’s a non-zero probability it reports an item is in the set when it’s actually not.

Burton H. Bloom

- The data structure was conceived by Burton H. Bloom in 1970.

Source of the exemple

- Theory and Practice of Bloom Filters for Distributed Systems
- Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz

Size constant

- It does not matter how many elements do we store the space will be the same.
- A bloom filter with 10^3 elements will take the same amount of space as a bloom filter with 10^32 elements and the same space as bloom filter with 0 elements.
- Trade of:
  - The more elements, the more uncertain is the "in the set answer".

Constant reponse time

- All the operations are taking constant time.

Exemple with alphabet letters

The set

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Size = 27 * Short = 432 bytes

Probalistic Data Structure

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Size = 2 bytes

Test if a letter is present in the set.

Rawid	Short	Short
1	A	0
2	B	0
3	C	1
4	D	0
5	E	0
6	F	1
7	G	1
8	H	1
9	I	0
10	J	1
11	K	0
12	L	0
13	M	1
14	N	0
15	O	0
16	P	1
17	Q	0
18	R	0
19	S	1
20	T	0
21	U	0
22	V	1
23	W	0
24	X	0
25	Y	0
26	Z	0

In the set

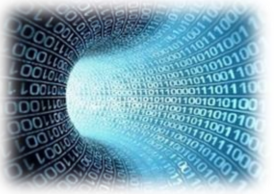
Size = 27 * Short * 2 = 864 bytes

Size exemples

binary-array of millions of elements

	N	bits	size byte	Mo
bits	1 000 000	8	125 000	125

		char per word		Char = 16 bits	size in bits	Mo	Go
words	1 000 000	6	6 000 000	16	96 000 000	96 000	96
			# chars				



Bit array

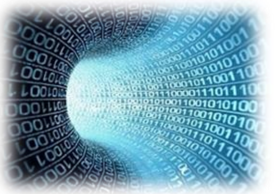
- A Bloom filter is an array of m bits for representing a set

- $S = \{x_1, x_2, \ldots, x_n\}$ of n elements

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

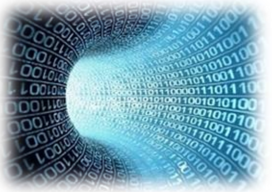
Hash functions

- The key idea is to use k hash functions,
  - $h_i(x)$, $1 \le i \le k$ to map items $x \in S$
  - To random numbers uniform in the range **1**, . . .m .
- The hash functions are assumed to be uniform.
- We write a hash function that, instead of a single hash code, produces k hash codes for a given object.



Multiple Hash Function

- If a Bloom filter returns that an item is member of the set, there's a certain probability for a false positive.



Inserting and testing

- An element $x \in S$ is inserted into the filter by setting the bits $h_i(x)$ to one , $1 \le i \le k$.

- Conversely, y is assumed a member of S if the bits $h_i(y)$ are set, and guaranteed not to be a member if any bit $h_i(y)$ is not set.

Exemple

- 16 bits map

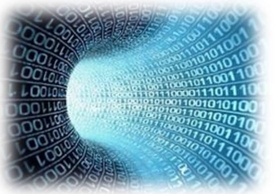
- Three hash functions are used: h_1 , h_2 , and h_3 , being MD5, SHA1 and CRC32, respectively.

- When adding an element, the values of h_1 through h_3 (modulo 16) are calculated for the element, and corresponding bit positions are set to one.

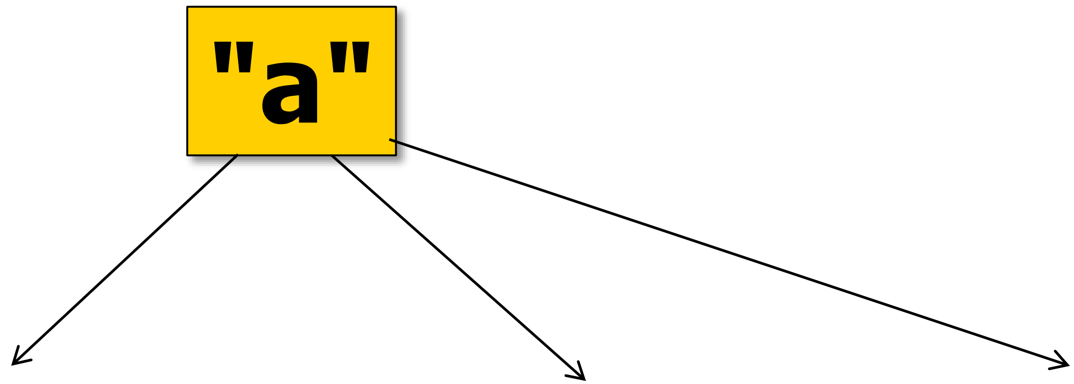
Steps

- Inserting a
- Inserting b
- Inserting l
- Inserting y

- Testing q
- Testing z



values of h_1 through h_3 (modulo 16)



Hash	MD5	SHA1	CRC32
hi	h1	h2	h3
a	0cc175b9c0f1b6a831c399e269772661	86f7e437faa5a7fce15d1ddcb9eaeaea377667b8	e8b7be43

values of h_1 through h_3 (modulo 16)

"a"

Hash	MD5	SHA1	CRC32
hi	h1	h2	h3
a	0cc175b9c0f1b6a831c399e269772661	86f7e437faa5a7fce15d1ddcb9eaeaea377667b8	e8b7be43

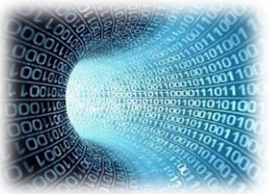
Modulo 16 ???

Binary Modulo Operation

$$(\text{number})_{base} \ \% \ \text{base}^{y}$$

$$\text{Last } y \text{ digits of } (\text{number})_{base}$$

Modulo 10^{y} base 10



X	P	mod
5	10	5
12	10	2
25	10	5
36	10	6
45	10	5
55	10	5
68	10	8
74	10	4
89	10	9
92	10	2

X	P	mod
68	100	68
105	100	5
257	100	57
396	100	96
458	100	58
589	100	89
623	100	23
758	100	58
899	100	99
985	100	85

X	P	mod
750	1000	750
1505	1000	505
2569	1000	569
3201	1000	201
4580	1000	580
5692	1000	692
6548	1000	548
7845	1000	845
8592	1000	592
9541	1000	541

Modulo 10^{y} base 10

X	P	mod
68	10	8
105	10	5
257	10	7
396	10	6
458	10	8
589	10	9
623	10	3
758	10	8
899	10	9
985	10	5

X	P	mod
750	100	50
1505	100	5
2569	100	69
3201	100	1
4580	100	80
5692	100	92
6548	100	48
7845	100	45
8592	100	92
9541	100	41

X	P	mod
1508	1000	508
10250	1000	250
25641	1000	641
36987	1000	987
45871	1000	871
51489	1000	489
65478	1000	478
73581	1000	581
82467	1000	467
98547	1000	547

Modulo 10^y base 10

10^y y = 1

X			P	mod
5	0	5	10	5
12	1	2	10	2
25	2	5	10	5
36	3	6	10	6
45	4	5	10	5
55	5	5	10	5
68	6	8	10	8
74	7	4	10	4
89	8	9	10	9
92	9	2	10	2

10^y y = 2

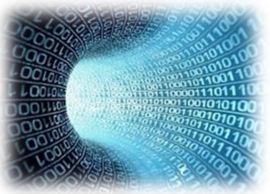
X			P	mod
68	0	68	100	68
105	1	05	100	5
257	2	57	100	57
396	3	96	100	96
458	4	58	100	58
589	5	89	100	89
623	6	23	100	23
758	7	58	100	58
899	8	99	100	99
985	9	85	100	85

10^y y = 3

X			P	mod
750	0	750	1000	750
1505	1	505	1000	505
2569	2	569	1000	569
3201	3	201	1000	201
4580	4	580	1000	580
5692	5	692	1000	692
6548	6	548	1000	548
7845	7	845	1000	845
8592	8	592	1000	592
9541	9	541	1000	541

Last y digits of $(\text{number})_{\text{base}}$

Binary Modulo Operation



Modulo 16

Digit											
7	6	5	4	3	2	1	0	Bin	Hex	Dec	Mod 16
1	0	0	0	1	0	0	1	0b10001001	0x89	137	9
0	1	0	0	0	1	0	0	0b01000100	0x44	68	4
0	0	1	0	0	0	1	0	0b00100010	0x22	34	2
0	0	0	1	0	0	0	1	0b00010001	0x11	17	1
0	0	0	0	1	0	0	0	0b00001000	0x8	8	8

$16 = 2^4$

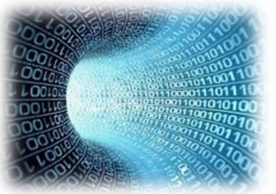
Last 4 digits

$y = 4$

Last y digits of $(\text{number})_{\text{base}}$

Emmanuel Fuchs

1 byte modulo



Last digits

Y

7	6	5	4	3	2	1	0						
128	64	32	16	8	4	2	1	Bin	Hex	Dec	Mod(x)	x	Power of 2
1	1	0	1	1	0	0	1	0b11011001	0xD9	217	1	2	1
0	0	0	0	0	0	0	1	0b00000001	0x01	1			
1	1	0	1	1	0	0	1	0b11011001	0xD9	217	1	4	2
0	0	0	0	0	0	0	1	0b00000001	0x01	1			
1	1	0	1	1	0	0	1	0b11011001	0xD9	217	1	8	3
0	0	0	0	0	0	0	1	0b00000001	0x01	1			
1	1	0	1	1	0	0	1	0b11011001	0xD9	217	9	16	4
0	0	0	0	1	0	0	1	0b00001001	0x09	9			
1	1	0	1	1	0	0	1	0b11011001	0xD9	217	25	32	5
0	0	0	1	1	0	0	1	0b00011001	0x19	25			
1	1	0	1	1	0	0	1	0b11011001	0xD9	217	25	64	6
0	0	0	1	1	0	0	1	0b00011001	0x19	25			
1	1	0	1	1	0	0	1	0b11011001	0xD9	217	89	128	7
0	1	0	1	1	0	0	1	0b01011001	0x59	89			
1	1	0	1	1	0	0	1	0b11011001	0xD9	217	217	256	8
1	1	0	1	1	0	0	1	0b11011001	0xD9	217			

2 bytes modulo

Last digits

	Digit																			
power	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Total			
val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	65535			
Bin	1	0	1	0	0	1	0	1	0	1	1	1	0	0	0	1	0b1010010101110001			
Dec	32768	0	8192	0	0	1024	0	256	0	64	32	16	0	0	0	1	42353		Mod(x)	x=□

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	4		
Bin	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0b0000000000000001	0x01		
Dec	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1		1	16

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	5		
Bin	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0b0000000000010001	0x011		
Dec	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	1	17		17	32

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	6		
Bin	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0b0000000000110001	0x031		
Dec	0	0	0	0	0	0	0	0	0	0	32	16	0	0	0	1	49		49	64

2 bytes modulo

Last digits

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	7		
Bin	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0b0000000001110001	0x071		
Dec	0	0	0	0	0	0	0	0	0	64	32	16	0	0	0	1	113		113	128

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	8		
Bin	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0b0000000001110001	0x071		
Dec	0	0	0	0	0	0	0	0	0	64	32	16	0	0	0	1	113		113	256

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	9		
Bin	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	1	0b0000000101110001	0x171		
Dec	0	0	0	0	0	0	0	256	0	64	32	16	0	0	0	1	369		369	512

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	10		
Bin	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	1	0b0000000101110001	0x171		
Dec	0	0	0	0	0	0	0	256	0	64	32	16	0	0	0	1	369		369	1024

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	11		
Bin	0	0	0	0	0	1	0	1	0	1	1	1	0	0	0	1	0b0000010101110001	0x571		
Dec	0	0	0	0	0	1024	0	256	0	64	32	16	0	0	0	1	1393		1393	2048

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	12		
Bin	0	0	0	0	0	1	0	1	0	1	1	1	0	0	0	1	0b0000010101110001	0x571		
Dec	0	0	0	0	0	1024	0	256	0	64	32	16	0	0	0	1	1393		1393	4096

2 bytes modulo

Last digits

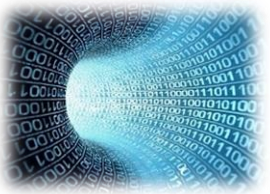
	Digit																			
power	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Total			
val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	65535			
Bin	1	0	1	0	0	1	0	1	0	1	1	1	0	0	0	1	0b1010010101110001			
Dec	32768	0	8192	0	0	1024	0	256	0	64	32	16	0	0	0	1	42353		Mod(x)	x=[illegible]
val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	13		
Bin	0	0	0	0	0	1	0	1	0	1	1	1	0	0	0	1	0b0000010101110001	0x571		
Dec	0	0	0	0	0	1024	0	256	0	64	32	16	0	0	0	1	1393		1393	8192

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	14		
Bin	0	0	1	0	0	1	0	1	0	1	1	1	0	0	0	1	0b0010010101110001	0x2571		
Dec	0	0	8192	0	0	1024	0	256	0	64	32	16	0	0	0	1	9585		9585	16384

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	15		
Bin	0	0	1	0	0	1	0	1	0	1	1	1	0	0	0	1	0b0010010101110001	0x2571		
Dec	0	0	8192	0	0	1024	0	256	0	64	32	16	0	0	0	1	9585		9585	32768

val	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	power of 2	16		
Bin	1	0	1	0	0	1	0	1	0	1	1	1	0	0	0	1	0b1010010101110001	0xA571		
Dec	32768	0	8192	0	0	1024	0	256	0	64	32	16	0	0	0	1	42353		42353	65536

values of h_1 through h_3 (modulo 16)



"a"

Hash	MD5	SHA1	CRC32
hi	h1	h2	h3
a	0cc175b9c0f1b6a831c399e269772661	86f7e437faa5a7fce15d1ddcb9eaeaea377667b8	e8b7be43
mod 16	1	8	3

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
h1	a	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h2	a	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
h3	a	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	^	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0

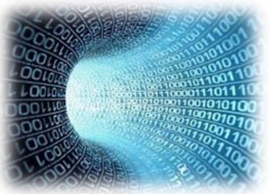
values of h_1 through h_3 (modulo 16)

"a"

Hash	MD5	SHA1	CRC32
hi	h1	h2	h3
a	0cc175b9c0f1b6a831c399e269772661	86f7e437faa5a7fce15d1ddcb9eaeaea377667b8	e8b7be43
mod 16	1	8	3

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
h1	a	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h2	a	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
h3	a	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	^	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0

values of h_1 through h_3 (modulo 16)



"b"

Hash	MD5	SHA1	CRC32
hi	h1	h2	h3
b	92eb5ffee6ae2fec3ad71c777531578f	e9d71f5ee7c92d6dc9e92ffdad17b8bd49418f98	71beeff9
mod 16	f	8	9

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
h1	b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
h2	b	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
h3	b	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	^	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1

values of h_1 through h_3 (modulo 16)

"b"

Hash	MD5	SHA1	CRC32
hi	h1	h2	h3
b	92eb5ffee6ae2fec3ad71c777531578f	e9d71f5ee7c92d6dc9e92ffdad17b8bd49418f98	71beeff9
mod 16	f	8	9

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
h1	b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
h2	b	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
h3	b	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	^	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1

A ^ b

a	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1
^	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	1
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Insert y

Hash	MD5	SHA1	CRC32
hi	h1	h2	h3
y	41529076959446 0e2e485922904f345d	95cb0bfd2977c761298d9624e4b4d4c72a39974a	fbdb2615
mod 16	d	a	5

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
h1	y	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
h2	y	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
h3	y	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	^	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0

Insert l

Hash	MD5	SHA1	CRC32
hi	h1	h2	h3
l	2db95e8e1a9267b7a1188556b2013b33	07c342be6e560e7f43842e2e21b774e61d85f047	9606c2fe
mod 16	3	7	e

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
h1	l	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
h2	l	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
h3	l	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	^	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0

Set map

y	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0
l	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0
^	0	0	0	1	0	1	0	1	0	0	1	0	0	1	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

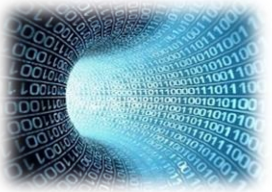
ab	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	1
yl	0	0	0	1	0	1	0	1	0	0	1	0	0	1	1	0
^	0	1	0	1	0	1	0	1	1	1	1	0	0	1	1	1
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Probe q and z

Hash	MD5	SHA1	CRC32
hi	h1	h2	h3
q	7694f4a66316e53c8cdd9d9954bd611d	22ea1c649c82946aa6e479e1ffd321e4a318b1b0	f500ae27
mod 16	d	0	7
z	fbade9e36a3f36d3d676c1b808451dd7	395df8f7c51f007019cb30201c49e884b46b92fa	62d277af
mod 16	7	a	f

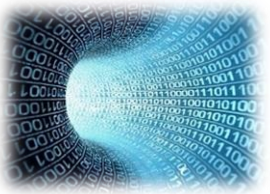
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	0	1	0	1	0	1	0	1	1	1	1	0	0	1	1	1	
q	1							1						1			Not in Set
z								1			1					1	In Set

False Positive



- The weak point of Bloom filters is the possibility for a false positive.
- False positives are elements that are not part of S but are reported being in the set by the filter.

Fault Positive

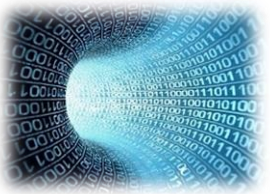


Collision

H(x) **H(y)**

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

False Positive



- If only a single hash function were used to indicate membership in the set, the probability of a false positive would be higher than using multiple hash functions.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	0	1	0	1	0	1	0	1	1	1	1	0	0	1	1	1	
q	1							1						1			Not in Set
z								1			1					1	In Set

False Positive

- If only a single hash function were used to indicate membership in the set, the probability of a false positive would be higher than using multiple hash functions.

Empty

Collision

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	0	1	0	1	0	1	0	1	1	1	1	0	0	1	1	1	
q	1							1						1			Not In Set
z								1			1					1	In Set