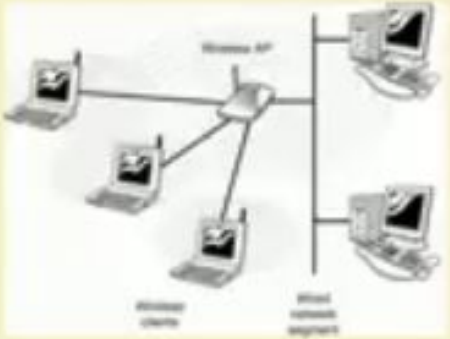
Bloom Filter

Probabilistic data structure

Querying

ISBN present in collection?

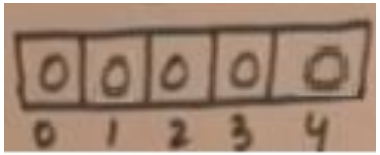


IP seen by switch?

10.0.21.102

$h1(x) = x \bmod 5$

$h2(x) = (2x+3) \bmod 5$



x	h1(x)	h2(x)	B
Insert 9	4	1	0 1 0 0 1 (0 1 2 3 4)
Insert 11	1	0	1 1 0 0 1 (0 1 2 3 4)

Query 15

$h_1(15) = 0$ $h_2(15) = 3$

SURELY 15 is not present

Query 16

$h_1(16) = 1$ $h_2(16) = 0$

FALSE POSITIVE

16 was probably present

How a Bloom Filter Works

- A *Bloom filter* is an array of bits, together with a number of hash functions.
- The argument of each hash function is a stream element, and it returns a position in the array.
- Initially, all bits are 0.
- When input x arrives, we set to 1 the bits h(x), for each hash function h.

Error types

- False Negative: answering “not there” on an element that is in the set.
  - Never happens for Bloom Filters
- False Positive: answering “is there” on an element that is not in the set
  - We design the filter so that the probability of a false positive is very small.

In hash table the object gets stored to the bucket(index position in the hash table) the hash function maps to.	Bloom filters doesn't store the associated object. It just tells whether it is there in the bloom filter or not.
Hash tables are less space efficient.	Bloom filters are more space efficient. it's size is even the less than the associated object which it is mapping.
Supports deletions.	It is not possible to delete elements from bloom filters.
Hashtables give accurate results.	Bloom filters have small false positive probability. (False positive means it might be in bloom filter but actually it is not.)

In a hashtable either we should implement multiple hash functions or have a strong hash function to minimize collisions.	A bloom filter uses many hash functions. There is no need to handle collisions.
Hashtables are used in compiler operations, programming languages(hash table based data structures),password verification, etc.	Bloom filters find application in network routers, in web browsers(to detect the malicious urls), in password checkers(to not a set a weak or guessable or list of forbidden passwords), etc.

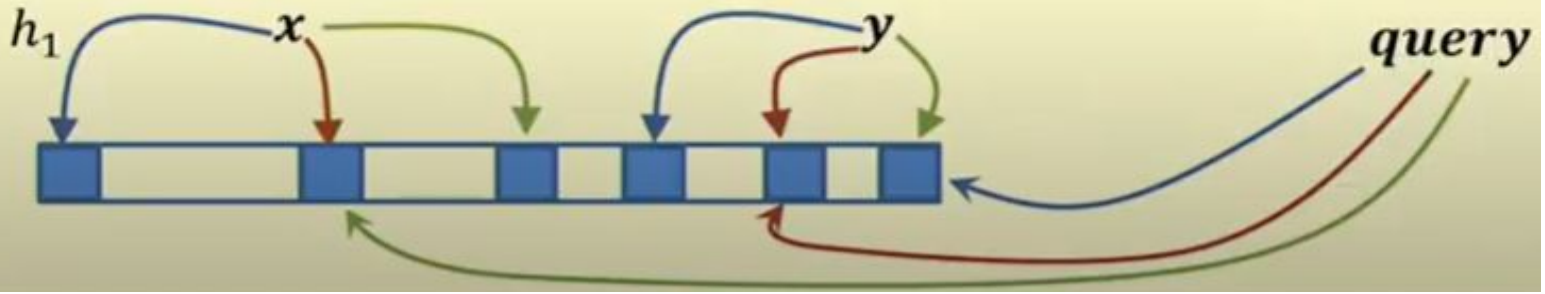
Classic Uses of BF: Data Bases

- **Join:** Combine two tables with a common domain into a single table. But it is costly operation in distributed DB.
- **Semi-join:** A join in distributed DBs in which only the joining attribute from one site is transmitted to the other site and used for selection. The selected records are sent back.
- **Bloom-join:** A semi-join where we send only a BF of the joining attribute.

Bloom Filter

- If the element x has been added to the Bloom filter, then $Lookup(B, x)$ always return PRESENT

- If x has not been added to the filter before?
  - $Lookup$ sometimes still return PRESENT



Designing Bloom Filter

- Want to minimize the probability that we return a false positive
- Parameters $m = |B|$ and $k =$ number of hash functions
- $k = 1 \Rightarrow$ normal bit-array

- What is effect of changing k?

Analysis of Bloom filter

Given m and n find optimal number of hash functions so that false positive rate is minimum.

Probability of a false positive

Total no of cells in a bloom filter = m

No. of hash functions = k

Probability that one cell is mapped to 1 when one element is inserted is 1/m

Probability that a cell is not mapped to 1 when one element is inserted is 1 – 1/m

Therefore 1 – 1/m is the probability that a cell is not mapped to 1 when one element is inserted by that hash function.

Probability of a false positive

But there are k hash functions.

Probability that a cell is not mapped to 1 when one element is inserted by k hash function is (1 – 1/m) (1 – 1/m) .... k times.

$$\left(1 - \frac{1}{m}\right)^k$$

But there are n elements to be hashed.

Probability that a cell is not mapped to 1 when n elements are inserted by k hash function is

$$\left(1-\frac{1}{m}\right)^k \quad \left(1-\frac{1}{m}\right)^k \quad \text{.... n times.}$$

$$\left(1-\frac{1}{m}\right)^{kn}$$

$$\left(1 - \frac{1}{m}\right)^m = e^{-1}$$

- $\Pr[h_i(x) = 0] = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$

When Lookup will return present? When all of the k bits $h_1(x), h_2(x) \ldots\ldots h_k(x)$ return 1.

$$\Pr[Lookup(B, x) = \text{PRESENT}) = \left(1 - e^{-kn/m}\right)^k$$

Can we choose k to minimize this probability

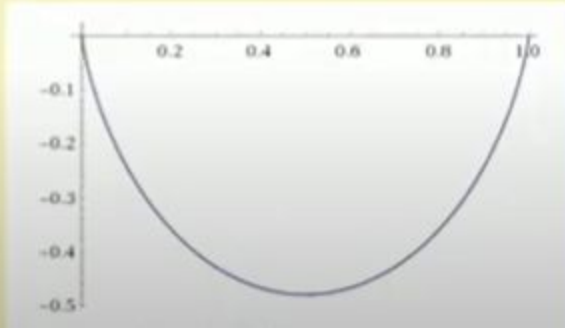
$p = e^{-kn/m}$ ⇨ $k = -\frac{m}{n}\log(p)$

Choosing number of hash functions

- $p = e^{-kn/m}$ ⇨ k = $-\frac{m}{n}\log(p)$
- Log (False Positive) =

$$\log(1 - p)^k = k\log(1-p) = -\frac{m}{n}\log(p)\log(1-p)$$

Minimized at $p = \frac{1}{2}$, i.e. $k = \frac{m}{n}\ln(2)$



minimized for $k = \ln 2 \times m/n$, in which case it becomes

$$\left(1 - e^{kn/m}\right)^k = \left(\frac{1}{2}\right)^k = (0.6185)^{m/n}$$

$$\left(\frac{1}{2}\right)^{k} = (0.6185)^{m/n}.$$

False positive rate under various m/n and k combinations.

m/n	k	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8
2	1.39	0.393	0.400						
3	2.08	0.283	0.237	0.253					
4	2.77	0.221	0.155	0.147	0.160				
5	3.46	0.181	0.109	0.092	0.092	0.101			
6	4.16	0.154	0.0804	0.0609	0.0561	0.0578	0.0638		
7	4.85	0.133	0.0618	0.0423	0.0359	0.0347	0.0364		
8	5.55	0.118	0.0489	0.0306	0.024	0.0217	0.0216	0.0229	
9	6.24	0.105	0.0397	0.0228	0.0166	0.0141	0.0133	0.0135	0.0145
10	6.93	0.0952	0.0329	0.0174	0.0118	0.00943	0.00844	0.00819	0.00846
11	7.62	0.0869	0.0276	0.0136	0.00864	0.0065	0.00552	0.00513	0.00509
12	8.32	0.08	0.0236	0.0108	0.00646	0.00459	0.00371	0.00329	0.00314
13	9.01	0.074	0.0203	0.00875	0.00492	0.00332	0.00255	0.00217	0.00199