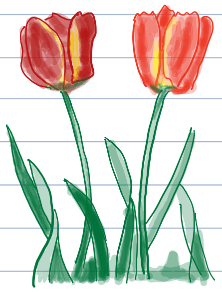
Secure Distributed Systems

CompSci 661 / 461



These notes:

- Bloom Filters

Bloom Filters (Harold Bloom, 1970)

A very popular data structure for probabilistically determining if an item x is an element of a set S, without transmitting/storing the full set S.

- the probability of a correct answer is inversely proportional to the size of the compact representation of S.

Some preliminary definitions:

The Bloom Filter has a chance of answering incorrectly. To evaluate/characterize its performance, we need to distinguish the ground truth from the result of asking the Bloom Filter.

ground truth
- Positive - $x \in S$. (x is in S)
- Negative - $x \notin S$ (x is not in S)

A guess is True if the guess is correct; and False if the guess is incorrect.

Hence, there are four possibilities

	Positive	Negative
True	TP	TN
False	FP	FN

E.g., FP is an incorrect guess that $x \in S$, when in fact $x \notin S$.

The False Positive rate (FPR) is

$$\frac{FP}{FP + TN} \qquad \begin{matrix} \leftarrow \text{number of False Positives} \\ \leftarrow \text{number of Negatives (all of them)} \end{matrix}$$

The False Negative rate (FNR) is

$$\frac{FN}{TP + FN} \qquad \begin{matrix} \leftarrow \text{number of False Negatives} \\ \leftarrow \text{number of Positives (all of them)} \end{matrix}$$

Bloom Filters have an

- Adjustable False Positive Rate (FPR)

- Zero False Negative Rate (FNR)

Let's see what's involved

Here's how Bloom Filters work.

① Create an array of m bits, all cleared (zeros)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

② For each element i of S, send it thru a hash function K times with prefixes $X_1, X_2, \dots, X_K$

$$h(x_1, i) \bmod m = h_1$$

$$h(x_2, i) \bmod m = h_2$$

$$\vdots$$

- we have K different hashes of the same input i

- h_1 is between 0 and $(m-1)$, of course.
- Set indices $h_1, h_2, \dots, h_K$ in the bit array.
- IF they are already set, Keep them set.

③ To test if $j \in S$: Send j thru the K hashes.

$h(x_1, j) \bmod m = h_1$ etc

Check each index in the array $(h_1, h_2 \dots h_K)$

- IF <u>any</u> are 0, there is zero chance j is in S.
- IF <u>all</u> are 1, we report that j is in S, but there is a non-zero chance we are wrong (False Positive)

In sum n items, m bits in array, K hash functions

http://www.eecs.harvard.edu/~michaelm/postscripts/im2005b.pdf

Example.

Set $n = 16$ bits in array
$K = 2$
$b = 4$

0	0	0	0	0	0	0	0

insert 1 element "a". And say $h_1 = \text{hash}(x_1, \text{element}) \otimes 16 = 2$
$h_2 = \text{hash}(x_2, \text{element}) \otimes 16 = 5$

0	0	1	0	0	1	0	0

we might insert a few more elements ...

0	0	1	1	0	1	0	1

Now check if "a" is in our filter.

$$\left.\begin{matrix} h_1 = 2 \\ h_2 = 5 \end{matrix}\right\}$$ both are 1, therefore is (supposedly)

IF we check for "b" and find that

$$\left.\begin{matrix} h_1 = 2 \\ h_2 = \underline{1} \end{matrix}\right\}$$ Since at least one index is 0, we know that "b" is not in the original set.

how do we get a false positive? Say we never inserted "C" into the filter. But it just so happens that

$$\left.\begin{matrix} h_1 = 2 \\ h_2 = 3 \end{matrix}\right\}$$ Since both are unfortunately set, we have a false positive.

The false positive rate depends on the number of elements – not the length of each element.

What is the FPR of a Bloom Filter?

(from Wikipedia)

Let's derive the false positive rate.

- For each hash function, we set 1 index in the array

  The probability that one of the m bits is set is $(1/m)$

- Probability of not being set is $(1-1/M)$
- Prob of not being set by K hash functions: $(1-1/m)^K$

- Let $n = |S|$. Since we will insert n elements, the probability that an index of the m bit array is not set is $(1-\frac{1}{m})^{kn}$

- Probability that an index is set is: $1-(1-1/m)^{kn}$

The Probability that K indicies are set for an element that is not in S is : $\left(1-(1-\frac{1}{m})^{kn}\right)^K$

which is approximately:

$$p \approx \left(1-e^{\frac{-kn}{m}}\right)^k$$

How many hash functions should we use?

It has been shown that K is optimal for a desired FPR of P when $K = -\log_2(p)$

E.g., for $p = 0.01 \Rightarrow K = \lceil -\log_2(0.01) \rceil = \lceil 6.6 \rceil = 7$

How many hash functions should we use?

[long version]

From Mitzenmacher

We want the value of K that mimizes the FPR

So we take the derivative of the FPR w.r.t. K

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

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

$$= \exp\left(\ln\left(1 - \exp\left(-\frac{kn}{m}\right)\right)^K\right)$$

$$= \exp\left(K \ln\left(1 - \exp\left(-\frac{kn}{m}\right)\right)\right)$$

let $e^x = \exp(x)$ for clarity

since $a = \exp(\ln(a))$

Minimizing x also minimizes $\exp(x)$

let $g = K \ln\left(1 - e^{-kn/m}\right)$

$$\frac{\partial g}{\partial K} = \ln\left(1 - e^{-kn/m}\right) + \frac{kn}{m} \cdot \frac{e^{-kn/m}}{1 - e^{-kn/m}}$$

This derivative is equal to 0 when $k = \ln(2)\frac{m}{n}$ (not shown here)

Let's substitute our optimal value of K into our equation for the FPR, and solve for m

$k = \ln(2)\frac{m}{n}$

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

$$= \left(1 - e^{-\ln(2)\frac{m}{n}\frac{n}{m}}\right)^{\ln(2)\frac{m}{n}}$$

$$= \left(1 - e^{-\ln 2}\right)^{\ln(2)\frac{m}{n}}$$

$$= \left(1 - \frac{1}{2}\right)^{\ln(2)\frac{m}{n}}$$

$$p = \left(\frac{1}{2}\right)^{\ln(2)\frac{m}{n}}$$

$$\ln(p) = \ln\left(\left(\frac{1}{2}\right)^{\ln(2)\frac{m}{n}}\right)$$

$$= \ln(2)\frac{m}{n}\ln\left(\frac{1}{2}\right)$$

$$= \frac{m}{n}\ln(2)(-\ln(2))$$

$$\ln(p) = -\frac{m}{n}\ln^2(2)$$

$\ln^2(2) = \ln(2)\ln(2)$

$$\frac{n\ln p}{\ln^2(2)} = -m$$

$$m = \frac{-n\ln p}{\ln^2(2)}$$

$\leftarrow$ number of bits in array given desired FPR of p and n.

And given n and m, the optimal number of hash functions is $k = \frac{m}{n}\ln 2$. (not an integer!)

OR

Note that $\frac{m}{n} = \frac{-\ln p}{\ln^2(2)}$

and so $K = \frac{-\ln p}{\ln^2(2)} \cdot \ln 2 = \frac{-\ln p}{\ln 2} \Rightarrow K = -\log_2 p$