Bloom Filter

Probabilist data structure used by the crawlers to check if a URL has been already crawled or not.

Instead of hashing the URLs and check for collisions, which is too costly in space,

the idea of the Bloom filter is to create a binary array $B[1,m]$ and map each URL to k positions in B, using a family of k hash functions

$(h_i : U \to \{0, \dots, B-1\} \quad \forall i = 1, \dots, k)$.

If $B[h(x)]$ is set to zero, set it to one, otherwise do nothing.

Given a URL v

if $B[h_i(v)] == 1 \quad \forall i : 1, \dots, k$

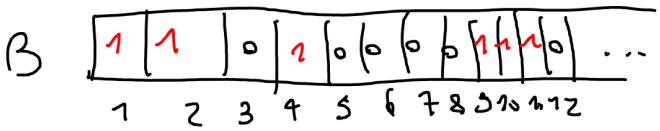
it means that we have already crawled v

As you can see, we don't need to store the whole URL, but only a sort of fingerprint.

However, we might have false positive

EXAMPLE FALSE POSITIVE

Given 3 hash function, a URL that we have yet to crawl (AAA) and a Bloom filter already populated by other crawled URLs



if $h_1(AAA) \wedge h_2(AAA) \wedge h_3(AAA)$ happens to be equal to a position set to 1, we will wrongly think that AAA was already crawled

What's the probability of false positive?

Assuming that the hashes are distributed perfectly at random (Simple Uniform Hashing)

$$p(B[h_i(x)] == 0) = \left(1 - \frac{1}{m}\right)^{k \cdot n} \simeq e^{\frac{-k \cdot n}{m}}$$

Where $i = 1, \ldots, k$ and x is taken from a set of size n

The probability of a false positive is

$$p(\forall i : B[h_i(x)] == 1) =$$

$$= p(B[h_i(x)] == 1)^k =$$

$$= \left(1 - p(B[h_i(x)] == 0)\right)^k \simeq$$

$$\simeq \left(1 - e^{\frac{-k \cdot n}{m}}\right)^k$$

THIS PROBABILITY IS BOUNDED BY

K : LARGE K, LESS FALSE POSITIVE

WHAT'S THE OPTIMAL VALUE FOR k?

By minimizing the previous probability:

$$\tilde{k} = \ln 2 \cdot \left(\frac{m}{n}\right)$$

Therefore the Bloom filter is good when:

$$\left(\frac{m}{n}\right) << \text{key length in bits} + \log n$$

SPECTRAL BLOOM FILTER

Variation of the Bloom filter, that uses an array of integers C instead of a binary one: each position counts the number of occurrences of an inserted item.

The space occupancy is larger and the error probability is the same, but it also permits deletion and lookup

Insertion(x)

```
for h in H:
    C[h(x)] ++
```

Deletion(x)

```
for h in H:
    C[h(x)] - -
```

Lookup(x)

return $\min\{C[h_1(x)], \ldots, C[h_k(x)]\}$

(minimum selection)