Bloom Filters

- Given a set $S = \{x_1, x_2, x_3, \ldots, x_n\}$ on a universe U, want to answer queries of the form:

  Is $y \in S$?

- Bloom filter provides an answer in
  - "Constant" time (to hash).
  - Small amount of space.
  - But with small probability of a false positive
    - Particularly useful when the answer is usually NO

1

Bloom Filters

Start with an m bit array, filled with 0s.

B | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0

Hash each item x_j in S k times. If $h_i(x_j) = a$, set $B[a] = 1$.

B | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0

To check if y is in S, check B at $h_i(y)$. All k values must be 1.

B | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0

Possible to have false positive; all k values are 1, but y is not in S.

B | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0

n items *$m = cn$ bits* *k hash functions*

2

False Positive Probability

n items *$m = cn$ bits* *k hash functions*

3

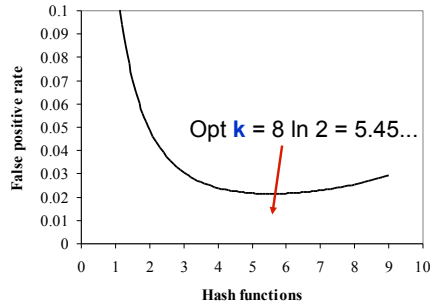
False Positive Probability

- Pr(specific bit of filter is 0) is
  $p' \equiv (1-1/m)^{kn} \approx e^{-kn/m} \equiv p \quad (p' \leq p)$
- If β is fraction of 0 bits in the filter then false positive probability for a new element is
  $(1-\beta)^k \approx (1-p')^k \approx (1-p')^k = (1-e^{-kn/m})^k$
- Find optimal at $k = (\ln 2)\, m/n$ by calculus.
  - So optimal false positive prob is about $(0.6185)^{m/n}$

n items *$m = cn$ bits* *k hash functions*

4

Graph of $(1-e^{-k/c})^k$ for c=8



m/n = 8

n items *$m = cn$ bits* *k hash functions*

5

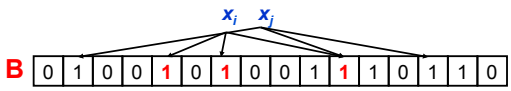
Applications

- Any scenario where space and efficiency are super important.
- Used a lot in networking
- Google BigTable uses Bloom filters to reduce the disk lookups for non-existent rows or columns.
  - Avoiding costly disk lookups considerably increases the performance of a database query operation

6

Handling Deletions

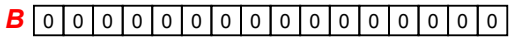
- Bloom filters can handle insertions, but not deletions.



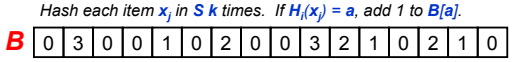
- If deleting x_i means resetting 1's to 0's, then deleting x_i will "delete" x_j.

Counting Bloom Filters

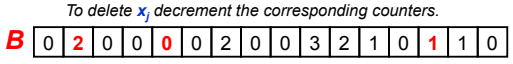
*Start with an **m** bit array, filled with 0s.*



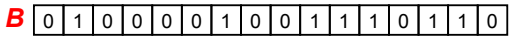
*Hash each item x_j in **S** **k** times. If $H_i(x_j) = a$, add 1 to **B[a]**.*



To delete x_j decrement the corresponding counters.



Can obtain a corresponding Bloom filter by reducing to 0/1.



Counting Bloom Filters: Overflow

- Must choose counters large enough to avoid overflow
  - e.g. for **c=8** choose **4** bits per counter
  - Average load using **k =** (ln **2**) **m/n** counters is ln **2**.
  - Probability a counter has load at least **16** is approximately $\mathbf{e^{-\ln 2}\,(\ln 2)^{16}/16!}$ which is roughly $\mathbf{6.78 \times 10^{-17}}$

Bloom filter numerous variations

- See papers on website.