CSC263 Monday Lecture (Guest Lecturer) Feb 5, 2018

BLOOM FILTERS

- invented by Bloom in the 70's
- probabalistic (or approximate) dictionaries

OVERVIEW

- stores F_S: summary/fingerprint of dynamic set S
  - regular dictionaries store actual elements
- BF_INSERT(F_S, x): $S := S \cup \{x\}$
- BF_SEARCH(F_S, x):
  - FALSE $\longrightarrow$ x is not in the set, $x \notin S$
  - TRUE $\longrightarrow$ x is probably (not certainly) in the set

- cannot delete & false positives when searching, so why use Bloom filter? Because it is very space efficient.

- "Ex: you are a webserver with a large list of blacklisted websites. Cannot store whole list in main memory, then store summary in BF. First check BF, if returns FALSE, know not blacklisted, only check whole list if BF returns TRUE (chance of false positive)."

- use when space constraints & false positives acceptable & when deleting not a big deal

Implementation of BLOOM FILTERS // params: m, t

- BF$[0, \ldots, m-1]$: an array of m bits, initialized to 0
- $\{h_1, \ldots, h_t\}$: set of $t < m$ hash functions where $h_i : U \to \{0, \ldots, m-1\}$,
  - recall all hash functions have independent uniform PDF

$\Theta(t)$ - BF_INSERT(BF, x): for i from 1 to t, BF$[h_i(x)] = 1$;
  - element produces "fingerprint" from hash functions which is stored

$\Theta(t)$ - BF_SEARCH(BF, x): for i from 1 to t, if (BF$[h_i(x)] == 0$) return false, else true

- params t, m affect false positive rate

FALSE POSITIVE ANALYSIS

- Assume n elements inserted. Compute probability that BF_SEARCH(BF, x) returns a false positive, if x has not been inserted.

- PROB[BF[$h_1(x)$]==1 && BF[$h_2(x)$]==1 && ... && BF[$h_t(x)$]==1]
  - h_i have uniform independent PDF over m, all equal probs.
- PROB[t. BF($h_i(x)$)==1] for any i from 1 to t
- PROB[BF[j] == 1] for j from 0 to $m-1$ after n insertions
  $\equiv$ 1 - PROB[BF[j] == 0] for j from 0 to $m-1$ after n insertions
  $\equiv 1 - (1 - 1/m)^{tn} = (1 - e^{-\frac{tn}{m}})^t$???