Bloom Filters

Djellel Difallah

Extra Lecture Notes

Set membership task

- S: set of elements
- x: an element
- Input (x, S)
- Output:
  - True if x in S
  - False if x not in S

➢ Maintain a list of elements of set S (in their original format)

➢ Scan the list and compare each element with x

Bloom Filter Motivation

- A set membership data structure
- pros: Fast and space efficient
- cons: Loss in accuracy

- Input (x, S)
- Output:
  - False (x not in S) Guaranteed always true
  - True (x in S) *Sometimes* this is false “False Positive”

Applications

- Spell checker
- Account creation
- ….
- Distributed DBMSs
- Caching

Bloom filter implementation

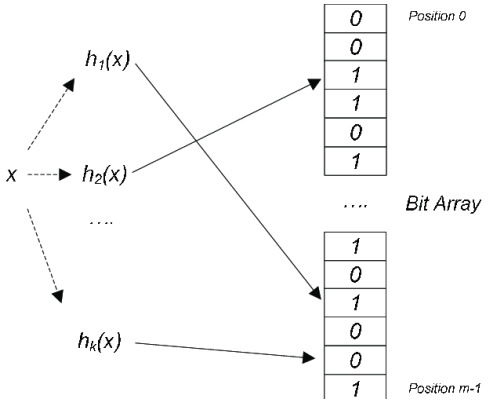
- We need two ingredients:
  - A bit array of size **m**
  - **k** hash functions h_1, h_2…., h_2

Bit Array of size **m**

0	0	0	0	0	0	0	0

- A hash function is any function that can be used to map data of arbitrary size onto data in a fixed domain
  - Deterministic
  - Different inputs may have similar outputs (collision)
  - Ideally: fast and uniform

- Operations
  - Add(x) to the bloom filter
  - Check(x) if x is in the bloom filter



False Positive Probability (FPP)

- We can control the error rate by setting:
  - m: Bit array size
  - k: Number of hash functions
  - n: number of expected elements

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

Demo
https://llimllib.github.io/bloomfilter-tutorial/