CS 7800: Advanced Algorithms

Spring 2026

1 Filters

Filters represent a set approximately, trading accuracy for space efficiency.

- Bloom Filters
- Quotient Filters
- Cuckoo Filters
- XOR Filters
- Ribbon Filters

1.1 Static vs Dynamic Filters

Static: The set of items is known in advance.

Dynamic: The set of items is not known in advance.

Dynamic filters typically support operations such as insertions, queries, and deletes.

1.2 Approximate Membership

Let S be a set and q be a query element.

A filter guarantees a false positive rate ε where $0 < \varepsilon < 1$.

- If $q \in S$, return **YES** with probability 1.
- If $q \notin S$, return
  - NO with probability $\geq 1 - \varepsilon$
  - YES with probability $\leq \varepsilon$

Filters therefore have **one-sided errors**:

$$\text{No false negatives}$$

1.3 Space Usage

Filters provide a lower bound on space:

$$\geq n \log \left(\frac{1}{\varepsilon} \right) \text{ bits}$$

Dictionary (hash table):

$$\Omega(n \log U) \text{ bits}$$

For most practical purposes:

$$\varepsilon = 2\% \Rightarrow \text{about 8 bits per item}$$

2 Filters in Databases

Disk accesses dominate database performance.

- RAM is much smaller than disk
- Filters store an approximate representation of keys in RAM
- Filters avoid disk accesses for negative queries

Typical operations required:

- Insert
- Query
- Delete

3 de Bruijn Graph

Consider alphabet $\{0, 1\}$ and strings of length k.

Vertices correspond to $(k-1)$-length strings.

An edge corresponds to a k-length string connecting:

$$\text{prefix}_{k-1} \rightarrow \text{suffix}_{k-1}$$

Example read:

$$\text{CACTGAA}$$

For $k = 4$, the k-mers are:

$$\{\text{CACT}, \text{ACTG}, \text{CTGA}, \text{TGAA}\}$$

These induce edges between $(k-1)$-mers.

3.1 Approximate Representation

A filter can represent a de Bruijn graph approximately.

Using filters for traversal introduces a small number of **topological errors**.

Question: How can we remove these errors?

4 Bloom Filters

A Bloom filter consists of:

- A bit vector of size m
- k hash functions

$$h_1, h_2, \ldots, h_k$$

To insert element x:

$$h_i(x) \rightarrow \text{bit positions}$$

All corresponding bits are set to 1.

4.1 Space

$$\approx 1.44\, n \log\left(\frac{1}{\varepsilon}\right) \text{ bits}$$

Bloom filters do **not support deletions**.

4.2 Questions

- How can we support deletions in Bloom filters?
- Can we merge two Bloom filters?

Counting Bloom Filters allow deletions but may introduce false negatives.

5 False Positive Analysis

Parameters:

$$m, k$$

Probability a bit is not set by one hash:

$$1 - \frac{1}{m}$$

Probability a bit is not set by any of k hashes:

$$\left(1 - \frac{1}{m}\right)^k$$

After n insertions:

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

Using the limit:

$$\lim_{m\to\infty} \left(1 - \frac{1}{m}\right)^m = e^{-1}$$

We obtain:

$$P(\text{bit is } 0) \approx e^{-kn/m}$$

$$P(\text{bit is } 1) = 1 - e^{-kn/m}$$

False positive probability:

$$(1 - e^{-kn/m})^k$$

6 Optimal Parameters

Increasing space decreases false positives.

Adding more elements increases false positives.

Optimal number of hash functions:

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

False positive rate:

$$\varepsilon \approx 2^{-k}$$

6.1 Example

Database size:

$$6 \text{ TB}$$

Keys of size:

$$512 \text{ B}$$

Number of keys:

$$\approx 12.88 \text{ billion}$$

Desired false positive rate:

$$1.56\%$$

We need:

- $k = 6$ hash functions
- ≈ 1 byte per key
- Total memory ≈ 12 GB

7 Single Hashing Filters

Use one hash function to compute a p-bit fingerprint.

$$x \to h(x)$$

Store fingerprints in a hash table.
False positives occur when two elements collide:

$$h(x) = h(y)$$

Probability:

$$P(x, y \text{ collide}) = \frac{1}{2^p}$$

8 Quotient Filters

Split hash value:

$$h(x) = b(x) \mid\mid t(x)$$

- $b(x)$: bucket index
- $t(x)$: fingerprint stored in table

Collisions handled with:

- Linear probing
- Robin Hood hashing

Space:

$$\approx n \log\left(\frac{1}{\varepsilon}\right) + 2.125n$$

False positive rate:

$$\frac{1}{2^r}$$

9 Cuckoo Filters

Based on cuckoo hashing.

Each item can be stored in one of two locations:

$$h_0(x), h_1(x)$$

Procedure:

1. Compute $h_0(x)$ and $h_1(x)$
2. Insert into empty slot
3. Kick out existing element if necessary

Typically:

$$b = 4$$

Space:

$$\approx n \log\left(\frac{1}{\varepsilon}\right) + 3n$$

False positive rate:

$$\approx \frac{2b}{2^f}$$