Project A3 — Approximate Membership Filters (XOR vs Cuckoo vs Quotient)

Approximate membership structures answer “is x probably in the set?” far faster and smaller than exact indexes. Modern designs—**XOR filter**, **Cuckoo filter**, and **Quotient filter**—extend classic Bloom filters with better space/speed trade-offs and, in some cases, **deletions** and more cache-friendly access. This project implements all three, builds a solid baseline (**blocked Bloom**), and maps when each design wins.

Learning Goals (What your experiments must reveal)

- Implement and validate **XOR**, **Cuckoo**, and **Quotient** filters (and a **blocked Bloom** baseline).
- Quantify **bits per entry (BPE)** vs **false positive rate (FPR)** and **lookup/insert/delete throughput**.
- Characterize **tail latency (p95/p99)** and **negative-lookup heavy** workloads.
- Understand **dynamic vs static** trade-offs (XOR is static; Cuckoo/Quotient support online inserts/deletes).
- Tie observations to **cache/TLB/branch** behavior and **load factor** effects.

Tools You Use

- **C/C++** (GCC/Clang) with `-O3 -march=native` (or NEON on ARM).
- A fast non-cryptographic hash (e.g., **xxHash**/Murmur3) with distinct seeds.
- `perf stat` (or similar) for uarch counters; timing harness & plotting scripts.

Scope: In-memory filters. Implement the core data structures **yourself** (no third-party libraries). SIMD is encouraged for bucket probes and small scans.

Scope & Requirements

- **XOR filter (static):** Build-time construction over a fixed key set; three hash positions; compact fingerprint array (8–16 bits). Must support **build** + **query**.
- **Cuckoo filter (dynamic):** Bucketized table (e.g., 2 choices, bucket size k=4), **fingerprints** per slot, **insert**, **query**, **delete**, bounded evictions, optional small **stash**. Measure behavior near high load factors.
- **Quotient filter (dynamic):** Single array with quotient/remainder; metadata bits (occupied, continuation, shifted). Support **insert**, **query**, **delete**. Emphasize contiguous, cache-friendly scans.
- **Blocked Bloom baseline:** Cache-line-blocked bitset; Supports **insert** + **query** (no deletes).

All structures use **64-bit keys** and a common API for apples-to-apples comparisons.

Experimental Knobs (orthogonal axes)

- **Set size (n):** 1M, 5M, 10M (scale up if RAM allows).
- **Target FPRs:** 5%, 1%, 0.1% (choose fingerprint sizes / table sizes accordingly).
- **Workloads:**
  - Read-mostly: 95% queries / 5% inserts (for dynamic filters).
  - Balanced: 50/50.
  - Read-only: 100% queries (all).
  - Negative-lookup share: 0%, 50%, 90%.

- **Load factor (dynamic only):** Sweep 0.4 → 0.95 (step 0.05).
- **Fingerprint width:** 8, 12, 16 bits (Cuckoo/XOR).

- **Threads:** 1, 2, 4, 8, … to physical cores (pinned)

Required Experiments & Plots

1. **Space vs accuracy:**
    - For each structure at the three target FPRs (false positive rate), measure **bits per entry** (including metadata) and achieved FPR on an independent negative set.
    - Show theory lines for Bloom (optional) to contextualize.

2. **Lookup throughput & tails:**
    - Queries/second vs negative-lookup share (0→90%).
    - Report **p50/p95/p99** latency for each structure and FPR.

3. **Insert/delete throughput (dynamic):**
    - Cuckoo & Quotient: ops/s across load factors; record **insertion failure rate** (cuckoo) and average probe length.
    - Cuckoo: bounded evictions, stash hits; Quotient: cluster length histograms.

4. **Thread scaling:**
    - Throughput vs threads for read-mostly and balanced mixes; document concurrency control (per-bucket locks, striped locks, or lock-free lookups) and contention points.

Reporting & Deliverables (commit everything to GitHub)

- **Source code** for XOR, Cuckoo, Quotient filters, plus blocked Bloom; uniform CLI and benchmark harness; plotting scripts.
- **Setup/methodology:** CPU/ISA, compiler & flags, OS, governor/SMT, pinning, hash choices/seeds, dataset generation.
- **Plots/tables** with units and error bars (≥3 runs). Clear labels showing fixed vs varied knobs.
- **Analysis** connecting BPE/FPR/throughput to structure internals (fingerprints, clusters, bucket size), counters, and load factor.
- **Limitations/anomalies** (e.g., rebuilds for XOR, stash growth, long quotient clusters) with hypotheses and mitigations.

Grading Rubric (Total 180 pts)

Implementation (70)

- (20) Correct **XOR filter** build & query; verifies target FPR.
- (20) Correct **Cuckoo filter** with insert/delete, bounded evictions, stash.
- (20) Correct **Quotient filter** with insert/delete (metadata handling).
- (10) **Blocked Bloom** baseline and common benchmark harness.

Experiment Design (40)

- (20) Complete sweeps (FPRs, load factor, negative rate, threads, distributions).

- (10) Fair, apples-to-apples parameters and independent test sets.
- (10) Tail-latency and cold/warm methodology.

5. **Performance Results (40)**

- (15) Clear **BPE vs FPR** and **throughput vs negative-rate** plots; error bars.
- (15) Insert/delete throughput vs load factor; failure/cluster statistics.
- (10) Thread scaling plots with discussion.

6. **Reporting Quality (30)**

- (30) Clean repo, reproducible scripts, labeled plots, and practical conclusions (when to choose which filter).