HES-SO

School of Engineering and Architecture

Data Structures & Algorithms — Lab 4

Benchmarking Hash Table Collision Strategies

Authors:
A. Fischer, M. Roth

Supervisors:
Dr. L. Girard

Fribourg, August 27, 2026

Contents

1 Objective **3**

2 Method **4**
2.1 Implementation 4
2.2 Load generation 4

3 Results **5**
3.1 Uniform keys 5
3.2 Skewed keys 5

4 Discussion **6**

5 Conclusion **7**

1 Objective

This report benchmarks three collision-resolution strategies for a fixed-capacity hash table — separate chaining, linear probing, and Robin Hood open addressing — under a uniform load factor and a skewed key distribution. The goal is to determine which strategy degrades least gracefully as the load factor $\alpha = n/m$ approaches 1, where n is the number of stored keys and m is the table capacity.

2 Method

2.1 Implementation

Each strategy was implemented as a template class over a fixed capacity $m = 2^{16}$, using the same 64-bit multiplicative hash function (Fibonacci hashing, $h(k) = (k \cdot \phi) \bmod 2^{64}$ shifted into the table's index range) so that any difference in measured performance is attributable to the collision strategy alone and not to hash quality.

Listing 2.1: Fibonacci hash used across all three strategies

```
uint64_t fib_hash(uint64_t key, unsigned bits) {
    constexpr uint64_t PHI = 0x9E3779B97F4A7C15ULL;
    return (key * PHI) >> (64 - bits);
}
```

2.2 Load generation

Two key distributions were tested:

- **Uniform** — keys drawn uniformly from $[0, 2^{32})$, simulating a well-behaved workload.
- **Skewed** — keys drawn from a Zipf distribution with $s = 1.2$, simulating a workload dominated by a small number of hot keys, common in cache and symbol-table use.

For each distribution, the table was filled to load factors $\alpha \in \{0.5, 0.7, 0.9, 0.95, 0.99\}$ and 1,000,000 successful lookups were timed per configuration, repeated 5 times with the median reported to reduce scheduler noise.

3 Results

3.1 Uniform keys

Table 3.1: Median lookup time (ns) under uniform key distribution

Strategy	$\alpha = 0.5$	0.7	0.9	0.95	0.99
Chaining	24	27	33	38	47
Linear probing	19	25	61	118	402
Robin Hood	21	24	34	42	71

Linear probing is fastest at low load but degrades sharply past $\alpha = 0.9$ as primary clustering lengthens probe sequences. Chaining degrades the least because bucket lookup cost is independent of global load factor; Robin Hood tracks chaining closely while keeping data in a single contiguous array, avoiding chaining's per-node allocation overhead.

3.2 Skewed keys

Table 3.2: Median lookup time (ns) under Zipf(s=1.2) key distribution

Strategy	$\alpha = 0.5$	0.7	0.9	0.95	0.99
Chaining	22	26	31	35	44
Linear probing	18	23	55	96	311
Robin Hood	19	22	29	33	52

The skewed workload is uniformly *faster* across all three strategies than the uniform one, at every load factor. This is a cache effect: hot keys land in a small, frequently-reused set of cache lines, so most of the 1M lookups hit L1/L2 rather than main memory, independent of collision strategy.

4 Discussion

Robin Hood hashing is the best overall choice for this workload range: it matches chaining's graceful degradation at high load while retaining linear probing's cache-friendly contiguous layout at low load, at the cost of a swap during insertion that chaining and plain linear probing do not pay. Linear probing should only be preferred when the load factor is bounded well below 0.9 in the deployment, since its degradation past that point is severe enough (4–9× uniform-key slowdown per doubling of the remaining headroom) to dominate any constant-factor advantage it holds at low load.

A limitation of this benchmark is that it measures successful lookups only; insertion cost and unsuccessful-lookup cost (relevant for set-membership use cases) were out of scope and are natural follow-up measurements.

5 Conclusion

Robin Hood hashing offers the best load-factor robustness of the three strategies tested, at a modest implementation complexity cost over chaining. Linear probing remains attractive only under a load-factor ceiling around $\alpha \leq 0.7$, past which its lookup cost grows far faster than either alternative.