CS 7800/4810: Advanced Algorithms Spring 2026

Lecture 13 — February 25, 2026

Prof. Prashant Pandey *Scribe: Daniel Long*

1 Overview

In the last lecture we discussed several different hash table designs.

- **Chaining.** Each table slot stores a (dynamic) container of keys mapping to that slot (typically a linked list or a small dynamic array).
- **Probing** Keys live inside the table array; collisions are resolved by probing.
  - **Linear probing:** probe sequence $h(x), h(x)+1, h(x)+2, \ldots$
  - **Quadratic probing:** probe sequence $h(x)+1^2, h(x)+2^2, \ldots$
- **Two-choice hashing.** Each key x has two candidate bins given by independent hash functions $h_0(x), h_1(x)$. Inserts always target the less-loaded bin. If both bins for an item are full, we declare the insertion as having failed.
- **Cuckoo hashing.** Each key has two candidate locations; on insertion failures we "kick out" an occupant and reinsert it, repeating for up to 500 steps (if no space is found after 500, declare the table to be full). Queries always check a constant number of spaces (2).

1.1 Stability and associativity

A useful way to reason about hash tables is via two high-level properties:

Definition 1 (Stability)**.** *A hash table is **stable** if, once a key is inserted, its physical location does not change except during global events like resizing/rebuilding.*

Definition 2 (Associativity)**.** *A hash table has **low associativity** if each key can reside in only a small number of possible locations/bins.*

- Stability often correlates with fast insertions and simpler concurrency control (less moving/relocation).
- Low associativity often correlates with fast queries/deletions (few candidate places to check).
- **Space efficiency** is a third axis, mostly controlled by how close the load factor α can approach 1.

We can apply this analysis to our previously studied hash tables:

- **Chaining:** stable (keys do not move after being appended), but *high associativity* within a chain (a key could be anywhere in the chain).
- **Linear/quadratic probing:** not stable (keys may shift under deletions/rehashing), and associativity can be effectively large (a key can drift far from $h(x)$).
- **Two-choice with bounded bins:** stable and low associativity (a key belongs to one of two bins).
- **Cuckoo hashing:** low associativity (constant candidates), but not stable (kick-outs move keys).

Today's topic is **Iceberg hashing**, a design that simultaneously achieves:

- **Stability** (no key movement except resizing)
- **Low associativity** (few bins to inspect)
- **High space efficiency** (load factor $1 - o(1)$ in theory)
- **Cache efficiency** (few cache-line touches per operation)

.

2 Iceberg hashing

2.1 Frontyard/backyard

This is a general idea that we can split the data into two places: the frontyard, which contains most of the data, and the backyard, which has a small number of overflowing items.

- **Frontyard:** a large primary table designed for speed and locality, where almost all keys reside.
- **Backyard:** a much smaller auxiliary structure that holds the small fraction of problematic keys (overflows).
- Optionally a **third level** (tiny stash) to handle extremely rare double-overflow cases.

2.2 Why not just increase bucket size?

The frontyard slots are divided into *buckets* (bins), and a key design decision is the *bucket size*.

Consider a bucketed two-choice table with m bins and bin capacity b (so total capacity mb). If we make b large (e.g. $b = \omega(\log n)$) then it becomes overwhelmingly likely that no bin overflows even at very high load. However, large b harms associativity (each bin scan becomes more expensive).

Iceberg hashing instead keeps bins small (for associativity/cache), and handles the remaining overflows in the backyard.

3 Revisiting balls-n-bins

3.1 Two-choice hashing properties

In the classic regime of placing n balls into n bins with two random choices per ball (always place into the less-loaded choice), the maximum load is $O(\log \log n)$ with high probability [3].

3.2 Iceberg lemma (overflow is small)

Iceberg hashing relies on a special phenomenon: even with *one-choice* hashing into bins of capacity slightly above the average, the *number of overflowed balls* can be bounded with high probability.

If we hash n items into $m = n/h$ bins (so the average is h per bin) and allow each bin capacity h, then the number of items that cannot fit in their designated bin is at most $m/\text{poly}(h)$ with high probability. Concretely, taking $h = \Theta(\log n)$ yields an overflow count of about $O(n/\log n)$ with high probability.

This "Iceberg lemma" (and stronger variants) is proved in the cited papers [1, 2].

3.3 Backyard sizing and deletions

A recurring issue in analyzing "always choose the less-loaded bin" with deletions is that the process is no longer monotone, since loads can decrease.

Iceberg hashing uses a structural advantage: the backyard is sized so that its *average occupancy stays constant*. Using the overflow bound above, if the overflow population is $O(n/\log n)$ and we allocate $\Theta(n/\log n)$ backyard bins, then the *average* backyard load is $O(1)$.

In this constant-average regime, we can apply two-choice load-balancing results to conclude that the *maximum* backyard load is $O(\log \log n)$ with high probability [3]. This is the key observation behind the Iceberg design: concentrate the hard cases into a small auxiliary table where the load remains well-controlled.

4 Details

4.1 Frontyard layout and stability

A basic Iceberg hash table stores each key x in its designated frontyard bin $\text{bin}(x)$ unless $\text{bin}(x)$ is full [1].

Importantly, **frontyard keys never move**: inserts fill an empty slot in the bin, deletes clear it, and the per-bin metadata is updated in place [1]. This gives the table stability (outside of resizing).

4.2 Fingerprinting

- Each frontyard bin physically contains up to $\Theta(h)$ record slots.

- Alongside the record slots, the bin stores small metadata: a vacancy bitmap, counters, and a compact table keyed by *fingerprints* [1].

Let $\mathrm{fp}(x)$ be a short hash (a fingerprint) of key x (e.g. $m = 8$ bits in a practical design). The *slot index is stored implicitly* as the index/position of the entry for $\mathrm{fp}(x)$ in the bin's fingerprint metadata.

Lookup path. To look up key x:

1. Compute $\mathrm{bin}(x)$ and fingerprint $\mathrm{fp}(x)$.
2. In the bin fingerprint metadata, if $\mathrm{fp}(x)$ is absent, x is either in the backyard or absent.
3. If present, use the implied slot index to check the corresponding slot.
4. If not found/confirmed, consult the backyard table.

4.3 Fingerprint collision probability

If a bin contains r keys and fingerprints are m bits (uniformly random), then for a *fixed* query key x (independent of the stored keys),

$$\mathbb{P}[\exists\, y \text{ in bin with } \mathrm{fp}(y) = \mathrm{fp}(x)] \leq \frac{r}{2^m}.$$

This upper bound is a good approximation when $r \ll 2^m$.

For example, with $r = 64$ and $m = 8$, this gives at most $64/256 = 1/4$.

4.4 Backyard and (optional) third level

The backyard can be implemented using many constant-time hash table designs; a two-choice bucketed table is acceptable since it remains small and constant-average-load [1, 2].

At extremely high load factors (very close to 1), one may add a tiny third level to absorb rare double-overflow events [1].

5 Summary

Iceberg hashing uses a frontyard/backyard decomposition to combine:

- **Stability:** keys in the frontyard never move during normal operations,
- **Low associativity:** operations touch only the designated bin (plus the backyard if needed),
- **Space efficiency:** the frontyard can run at load factor $1 - o(1)$ in theory, with a very small backyard,
- **Cache efficiency:** compact per-bin tables enable a small number of cache-line accesses.

References

[1] Michael A. Bender, Alex Conway, Martin Farach-Colton, William Kuszmaul, and Guido Tagliavini. Iceberg hashing: Optimizing many hash-table criteria at once. *Journal of the ACM*, 70(6):40:1–40:51, 2023. [1]

[2] Prashant Pandey, Michael A. Bender, Alex Conway, Martin Farach-Colton, William Kuszmaul, Guido Tagliavini, and Rob Johnson. IcebergHT: High performance hash tables through stability and low associativity. *Proc. ACM Manag. Data*, 1(1):47:1–47:26, 2023. [2]

[3] Berthold Vöcking. How asymmetry helps load balancing. *Journal of the ACM*, 50(4):568–589, 2003. [3]