Hashing (Part 2)

CS 7800/4810 Lecture Notes

1 Perfect Hashing [FKS 1984]

1.1 Goal

We want to build a hash table with:

- $O(1)$ **worst-case** query time (not just expected!)
- $O(n)$ space
- $O(n)$ expected preprocessing time

1.2 The Problem with Single-Level Hashing

With a single hash table of size $m = \Theta(n)$ and universal hashing:

- Expected chain length is $O(1)$
- But worst-case chain length is $O\left(\frac{\log n}{\log \log n}\right)$ w.h.p.

We can't get $O(1)$ worst-case with one level.

1.3 The Key Insight: Two-Level Hashing

Idea: Use a first-level hash table, but instead of storing chains as linked lists, store each chain in its own **secondary hash table** that has **no collisions**.

Level 1 (Primary Hash Table):

- Hash n keys into $m = \Theta(n)$ slots using universal hash function h
- Let C_t = number of keys hashing to slot t
- We know $E[C_t] = \frac{n}{m} = O(1)$

Level 2 (Secondary Hash Tables):

- For each slot t with C_t keys, create a secondary hash table of size $s_t = \Theta(C_t^2)$

Why C_t^2? This is the key to making collisions unlikely!

1.4 Analysis: Expected Collisions in Secondary Table

For slot t with C_t keys, using a universal hash function h_t into a table of size s_t:

$$E[\#\text{ collisions in table } t] = \sum_{i<j,\text{ both in } C_t} \Pr\{h_t(x_i) = h_t(x_j)\}$$

There are $\binom{C_t}{2} = \frac{C_t(C_t-1)}{2}$ pairs of keys in slot t.
With universal hashing, each pair collides with probability $\leq \frac{1}{s_t}$:

$$E[\#\text{ collisions}] \leq \binom{C_t}{2} \cdot \frac{1}{s_t} = \frac{C_t(C_t-1)}{2} \cdot \frac{1}{\Theta(C_t^2)} = O(1)$$

If we set $s_t = c \cdot C_t^2$ for a suitable constant c, we get:

$$E[\#\text{ collisions}] \leq \frac{C_t^2}{2} \cdot \frac{1}{c \cdot C_t^2} = \frac{1}{2c}$$

By choosing c large enough (e.g., $c = 1$), we get $E[\#\text{ collisions}] \leq \frac{1}{2}$.

1.5 Using Markov's Inequality

Markov's Inequality

Let X be a non-negative random variable:

$$\Pr(X > \alpha \cdot E[X]) \leq \frac{1}{\alpha}$$

Let X = number of collisions. We have $E[X] \leq \frac{1}{2}$.
Setting $\alpha = 2$:

$$\Pr(X > 1) \leq \Pr\left(X > 2 \cdot \frac{1}{2}\right) \leq \frac{1}{2}$$

Since collisions are integers, $X > 1$ means $X \geq 1$, so:

$$\Pr(X \geq 1) \leq \frac{1}{2} \quad \Rightarrow \quad \Pr(X = 0) = \Pr(\text{no collisions}) \geq \frac{1}{2}$$

1.6 Building Collision-Free Secondary Tables

Algorithm for each secondary table:

1. Pick a random universal hash function h_t
2. Hash all C_t keys into the table of size $s_t = c \cdot C_t^2$
3. If any collision occurs, go back to step 1
4. If no collisions, we're done!

Expected number of trials: Since each trial succeeds with probability $\geq \frac{1}{2}$, the expected number of trials is $\leq 2 = O(1)$.

1.7 Space Analysis

Total space = (primary table) + (all secondary tables)

$$E[\text{space}] = m + \sum_{t=1}^{m} s_t = m + \sum_{t=1}^{m} \Theta(C_t^2)$$

We need to bound $\sum_t C_t^2$.
Claim: $E\left[\sum_t C_t^2\right] = O(n)$ when $m = \Theta(n)$.
Proof:

Step 1: Rewrite C_t^2 as a double sum

For each slot t, the chain length C_t counts how many keys hash to slot t:

$$C_t = \sum_{i:h(x_i)=t} 1 = |\{i : h(x_i) = t\}|$$

When we square this, we get:

$$C_t^2 = \left(\sum_{i:h(x_i)=t} 1 \right)^2$$

To expand this square, recall that for any sum $\left(\sum_i a_i\right)^2 = \sum_i \sum_j a_i a_j$.
Here each $a_i = 1$ if $h(x_i) = t$ and $a_i = 0$ otherwise. So:

$$C_t^2 = \sum_{i:h(x_i)=t} \sum_{j:h(x_j)=t} 1 \cdot 1 = \sum_{i,j:h(x_i)=t \text{ and } h(x_j)=t} 1$$

Interpretation: C_t^2 counts the number of *ordered pairs* (i, j) such that both x_i and x_j hash to slot t. This includes:

- Diagonal pairs where $i = j$ (there are C_t of these)
- Off-diagonal pairs where $i \neq j$ (there are $C_t(C_t - 1)$ of these)

Indeed: $C_t^2 = C_t + C_t(C_t - 1) = C_t^2$. ✓

Step 2: Sum over all slots

Now we sum C_t^2 over all slots t:

$$\sum_t C_t^2 = \sum_t \sum_{i,j:h(x_i)=h(x_j)=t} 1$$

Key insight: We can swap the order of summation! Instead of:

"For each slot t, count pairs (i, j) that both hash to t"

we can think of it as:

"For each pair (i, j), check if they hash to the same slot"

This gives us:

$$\sum_t C_t^2 = \sum_{i,j} \mathbf{1}[h(x_i) = h(x_j)]$$

where $\mathbf{1}[h(x_i) = h(x_j)]$ is the indicator that equals 1 if x_i and x_j hash to the same slot (any slot), and 0 otherwise.

Step 3: Split into diagonal and off-diagonal

Diagonal terms ($i = j$): There are n such terms, each contributing 1.

$$\sum_i \mathbf{1}[h(x_i) = h(x_i)] = n$$

Off-diagonal terms ($i \neq j$): There are $n(n-1)$ such pairs.

$$E\left[\sum_{i \neq j} \mathbf{1}[h(x_i) = h(x_j)]\right] = \sum_{i \neq j} \Pr\{h(x_i) = h(x_j)\} \leq n(n-1) \cdot \frac{1}{m}$$

For $m = \Theta(n)$:

$$\leq n(n-1) \cdot \frac{c}{n} = O(n)$$

Step 4: Combine

$$E\left[\sum_t C_t^2\right] = n + O(n) = O(n)$$

Therefore:

$$E[\text{space}] = m + O(n) = O(n)$$

1.8 Query Algorithm

To query key x:

1. Compute $t = h(x)$ — find which primary slot
2. Look up x in secondary table t at position $h_t(x)$
3. If found, return; otherwise, x is not in the set

Time: $O(1)$ — just two hash computations and two table lookups. **Deterministic**, not expected!

1.9 Summary

Property	**Value**
Query time	$O(1)$ **worst-case**
Space	$O(n)$ expected
Preprocessing	$O(n)$ expected
Hash functions	$1 + m$ (one primary + one per slot)

FKS perfect hashing shows you can achieve a **static dictionary** with $O(1)$ worst-case lookup and linear space. The only downside is that it doesn't support insertions/deletions efficiently.

2 Linear Probing

Linear probing has great cache performance.

Insert(x): Put x at first available slot $[h(x) + i] \mod m$.

Requirement: $m > (1 + \epsilon) \cdot n$ (not just $m = \Omega(n)$).

2.1 Results

The performance of linear probing depends heavily on the hash function used. All results assume table size $m = (1 + \epsilon)n$.

Hash Function	Expected Time/Op	Reference
Totally random	$O(1/\epsilon^2)$	[Knuth, 1962]
$O(\lg n)$-wise independent	$O(1/\epsilon^2)$	[Siegel, FOCS 1989]
5-wise independent	$O(1)$	[Pagh, Pagh, Ruzic, STOC 2007]
Pairwise independent	$\Omega(\lg n)$ (lower bound)	[Pagh, Pagh, Ruzic, STOC 2007]
Some 4-wise independent	$\Omega(\lg n)$ (lower bound)	[Pătraşcu & Thorup, ICALP 2010]
Simple tabulation	$O(1/\epsilon^2)$	[Pătraşcu & Thorup, STOC 2011]

Key insights:

- Totally random hashing gives $O(1/\epsilon^2)$ expected time per operation
- 5-wise independence is sufficient for $O(1)$ expected time
- Pairwise independence is **not** sufficient — can be $\Omega(\lg n)$
- Simple tabulation (only 3-independent!) surprisingly achieves $O(1/\epsilon^2)$

2.2 Quadratic Probing

$$h(k, i) = (h(k) + i^2) \mod p$$

2.3 Double Hashing

Given two hash functions h_1 and h_2:

$$\tilde{h}(k, i) = (h_1(k) + i \cdot h_2(k)) \mod m$$

3 Cuckoo Hashing [Pagh & Rodler, 2004]

- 2 tables of size $m > (1 + \epsilon) \cdot n$
- 2 hash functions ($g \to A$, $h \to B$)

Query(x): Check $A[g(x)]$ and $B[h(x)]$.
Insert(x):

1. Put in $A[g(x)]$ or $B[h(x)]$
2. If kicked out y from $A[g(y)]$: move to $B[h(y)]$
3. etc.
4. If stuck, rehash entire structure

Properties:

- $(2 + \epsilon)n$ space

- 2 deterministic probes for query
- Fully random or $\Theta(\lg n)$-wise independence $\Rightarrow$ $O(1)$ expected update and $O(1/n)$ failure probability (for construction on n keys)
- Some 6-wise independent hash functions fail w.h.p. even if $m = n^{1+\epsilon}$ [Cohen & Kane, 2009]
- Simple tabulation hashing $\Rightarrow$ fail with prob $\Theta(1/n^{1/3})$ [Pătraşcu & Thorup, STOC 2011]
- $\Rightarrow \Theta(n^{4/3})$ inserts OK

4 2-Choice Hashing

- Use two hash functions, adding the inserted item to the shorter list
- During queries, we search in both lists

Result: In expectation, the length of the longest chain is $\Theta(\lg \lg n)$.
Generally, for d hash functions, the longest chain is of length:

$$\Theta\left(\frac{\lg \lg n}{\lg d}\right)$$

5 Iceberg Hashing

5.1 Motivation: The Space Efficiency Problem

In standard hashing with chaining, we have a fundamental inefficiency:

	Avg Fill $= h$	**Max Fill** $= b$	**Space Eff.** $= b/h$
Standard	$O(1)$	$O\left(\frac{\lg n}{\lg \lg n}\right)$	$\Theta\left(\frac{\lg n}{\lg \lg n}\right) \gg 1$

The problem: Each bucket must be sized for the *worst case* (max fill b), but on average only has h items. The ratio b/h is the wasted space factor.
With standard hashing:

- Average fill: $h = O(1)$ items per bucket
- Maximum fill: $b = O\left(\frac{\lg n}{\lg \lg n}\right)$ w.h.p.
- Space efficiency: $\frac{b}{h} = O\left(\frac{\lg n}{\lg \lg n}\right)$ — very wasteful!

Goal: Achieve space efficiency close to 1 (i.e., $b/h \approx 1$).

5.2 Key Insight: Increase Average Fill

If we increase the average fill h, the max fill b doesn't grow as fast:

Avg Fill h	**Max Fill** b	**Space Eff.** b/h	**Why?**
$O(1)$	$O\left(\frac{\lg n}{\lg \lg n}\right)$	$O\left(\frac{\lg n}{\lg \lg n}\right)$	Balls & bins bound
$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$	$b = h + O(\sqrt{h \lg n})$
$\gg \lg n$	$h + O(\sqrt{h})$	$1 + o(1)$	Concentration

Explanation of each regime:

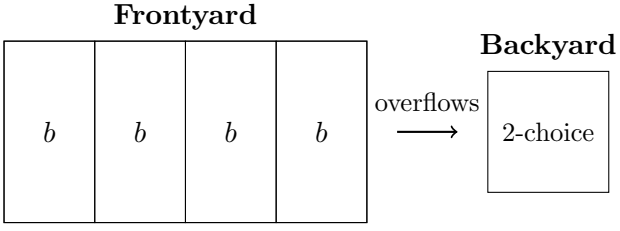
- **Row 1 ($h = O(1)$):** This is standard hashing with n balls in $m = \Theta(n)$ bins. Each bin gets $O(1)$ balls on average, but the maximum is $O\left(\frac{\lg n}{\lg \lg n}\right)$ by the balls-and-bins analysis. Space efficiency is poor.
- **Row 2 ($h = \Theta(\lg n)$):** Use $m = n/\lg n$ bins, so each bin gets $\lg n$ balls on average. By Chernoff bounds, the maximum load is $h + O(\sqrt{h \lg n}) = O(\lg n)$. Space efficiency becomes $\Theta(1)$ — a big improvement!
- **Row 3 ($h \gg \lg n$):** When average fill is very large, concentration is tight. The max fill is only $h + O(\sqrt{h})$ by standard deviation bounds. Space efficiency approaches $1 + o(1)$ — nearly optimal!

Intuition: When h is large, the law of large numbers kicks in. The number of balls in a bin concentrates around its mean h, with standard deviation $O(\sqrt{h})$. So the max fill is roughly $h + O(\sqrt{h})$, giving space efficiency:

$$\frac{b}{h} = \frac{h + O(\sqrt{h})}{h} = 1 + O\left(\frac{1}{\sqrt{h}}\right) \to 1$$

5.3 The Iceberg Structure

Idea: Frontyard and backyard hashing.



Structure:

- **Frontyard:** Array of m blocks, each of size b slots
- **Backyard:** Small hash table using 2-choice hashing for overflows

Parameters:

- n items total
- m blocks in frontyard
- b slots per block
- Expected items per block: $h = \frac{n}{m}$

5.4 Space Efficiency Analysis

Total space = bm (frontyard) + backyard size.
Space efficiency = $\frac{\text{total space}}{n} = \frac{bm}{n} = \frac{bm}{hm} = \frac{b}{h}$
Goal: Make b/h close to 1.

5.5 Two-Choice Hashing in the Backyard

The backyard uses 2-choice hashing to handle overflows efficiently.

Two-Choice Hashing Theorem (Berenbrink et al.

If you throw N balls into $N/\lg N$ bins using two-choice hashing (insert into the less loaded bin), the fullest bin will have:

$$\lg N + \lg\lg N + O(1) \text{ balls w.h.p.}$$

Comparison with single-choice:

- Single-choice: max load = $\Theta\left(\frac{\lg N}{\lg\lg N}\right)$ for N balls in N bins
- Two-choice: max load = $\lg\lg N + O(1)$ — exponentially better!

Note: This theorem does not hold with deletions, unless average bucket occupancy is $O(1)$.

5.6 Iceberg Theorem

Iceberg Theorem

If you throw N balls into $N/\lg N$ bins of size $\lg N + o(\lg N)$, the number of overflows will be $O(N/\lg N)$.

5.7 Proof Sketch

Setup:

- N items
- $m = N/\lg N$ blocks in frontyard
- Each block has size $b = \lg N + o(\lg N)$
- Average items per block: $h = \frac{N}{m} = \lg N$

Step 1: Bound overflows using concentration.

For a single block, the number of items X follows approximately a Poisson distribution with mean $\mu = \lg N$.

By Chernoff-type bounds, for $b = \lg N + c\sqrt{\lg N}$:

$$\Pr[X > b] = \Pr[X > \mu + c\sqrt{\mu}] \leq e^{-\Theta(c^2)}$$

Step 2: Expected number of overflows per block.

An overflow occurs when a block receives more than b items. The expected overflow from one block is small when b is slightly larger than h.

Step 3: Total overflows.

By linearity of expectation:

$$E[\text{total overflows}] = m \cdot E[\text{overflows per block}] = O(N/\lg N)$$

Step 4: Backyard handles overflows.

The $O(N/\lg N)$ overflows go to the backyard, which uses 2-choice hashing. Since the backyard has $O(N/\lg N)$ items with $O(1)$ average occupancy, it achieves $O(\lg \lg N)$ max load.

5.8 Putting It Together

The Iceberg Hashing Algorithm:

Insert(x):

1. Compute block $t = h_1(x) \mod m$
2. If block t has an empty slot, insert x there
3. Otherwise, insert x into the backyard using 2-choice hashing

Query(x):

1. Check block $h_1(x) \mod m$
2. If not found, check both possible backyard locations

5.9 Final Result

Property	**Value**
Space efficiency	$1 + O(1/\lg n)$
Query time	$O(1)$ expected
Insert time	$O(1)$ expected

Key insight: By using a frontyard with high average fill ($h = \lg n$) and a two-choice backyard for the few overflows, Iceberg hashing achieves near-optimal space efficiency while maintaining $O(1)$ operations.