# Complexity & Algorithms, Spring 2026

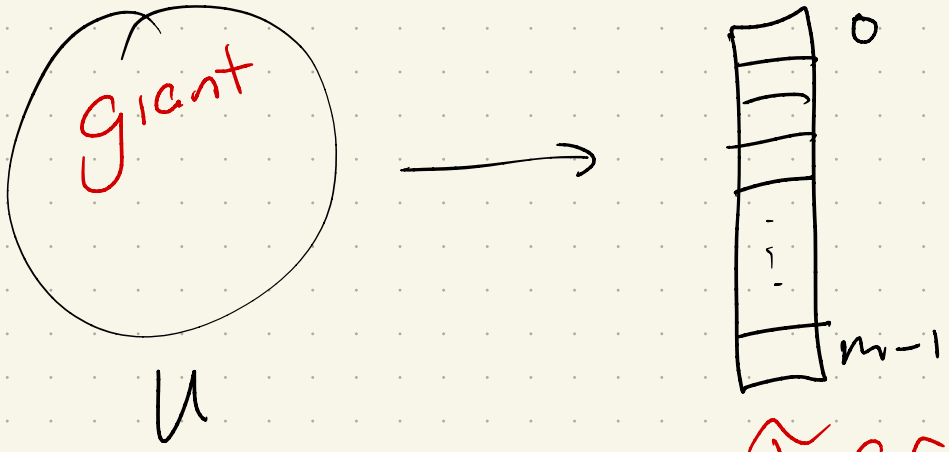
## Hashing

# Recap

- Final exam: Monday of finals week
- HW due next week
  - 2 (?) more HWs
    - one more essay-style
    - last HW - oral grading

# Hashing

function $h: \{0, \ldots, U-1\} \longrightarrow \{0, \ldots, m-1\}$




Problem:

Assumption: $U$ is huge

↳ but we won't use all of it!

Example from intro programming: dict in Python

Films ⟶ directors

h("StarWars") = "George Lucas"

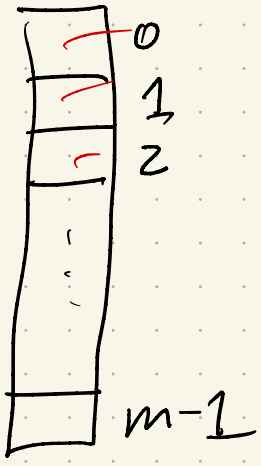
Natural idea: totally uniform

$$\Pr_{h \in \mathcal{H}}\{h(x) = t\} = \frac{1}{m}$$

$0 \le t \le m-1$

for some family $\mathcal{H}$ of functions

Table:




Note: Not avoiding collisions!

Problem:

Let $\mathcal{H} = \{f_i : f_i(x) = i \ \forall x \in \mathcal{U}$ & $\forall i \in \{0, \ldots, m-1\}\}$

$f_0(x) = 0$

$f_1(x) = 1$

$\vdots$

$\Pr_{f_i \in \mathcal{H}}\{h(x) = t\} = \frac{1}{m}$

Universal hashing: Minimize collisions:

$$\Pr_{h \in \mathcal{H}} \{ h(x) = h(y) \} = O\left(\frac{1}{m}\right) \quad \text{for all } x \neq y \in \mathcal{U}$$

$= \frac{1}{m}$ (strong)

$\leq \frac{c}{m}$

When do we get this?

→ $h(x) = [(ax) \bmod p] \bmod m$ for $0 < a < p$

p big, m smaller

(Confusingly, this is called the "uniform hashing assumption")

k-uniform: Maybe we want to avoid larger sets of collisions:

$$\Pr_{h \in \mathcal{H}}\left[h(x_1) \neq h(x_2) \neq \cdots \neq h(x_k)\right] = \frac{1}{m^k}$$

$= 1$

This is stronger! Why? Fix $p$.

Let $\mathcal{H} = \{ h_{a,b}(x) = \overline{ax+b} \pmod p,$
$a, b \in \mathbb{Z}_p, a \neq 0 \}$

$\{0, \ldots, p-1\}$

"affine hashing":

very common

This is universal: If $U = p$ and

if $x \neq y$, ~~then~~ and

$$h_{a,b}(x) = h_{a,b}(y)$$

$$\iff ax + b \pmod p = ay + b \pmod p$$

$$\iff a(x-y) \equiv 0 \bmod p$$

$$= k \cdot p^1$$

But: $p$ is prime

$\Rightarrow$ either $a$ or $x-y$ must be 0

not true → close

$a \neq 0$

by assumption

This is not 3-uniform:

Linear function $ax+b$ ← line

$\Rightarrow$ $h(x_3)$ = function of $h(x_1), h(x_2)$ ← on line

$$= h(x_1) + \frac{x_3 - x_1}{x_2 - x_1}\left(h(x_2) - h(x_1)\right)$$

(trust me)

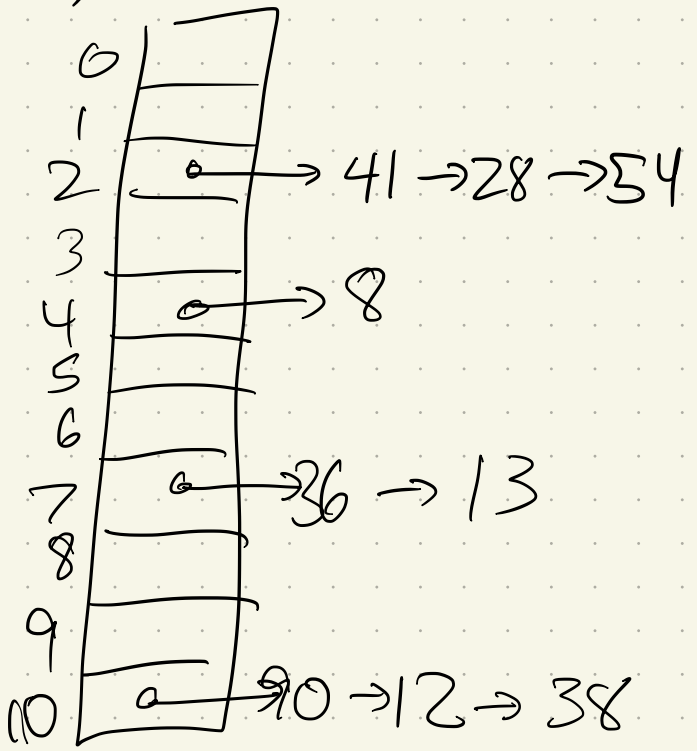
In other words:

Fix $h(x_1)$ & $h(x_2)$

$\Rightarrow$ uniquely determined $h(x_3)$ already.

# Chaining: Simple strategy

If $h(x) = h(y)$, store both at this spot.




Let $\ell(x)$ = length of list at $h(x)$

Time: $O(1 + \ell(x))$

How to analyze $\ell(x)$?

expected length

Indicator variable: (random var)

$$C_{x,y} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$$

Then, expected value of $C_{x,y}$:

$$E[C_{x,y}] = 1 \cdot Pr[C_{x,y} = 1] + 0 \cdot Pr[C_{x,y} = 0]$$

$$= Pr[C_{x,y} = 1] = \frac{1}{m}$$

if uniform hashing

Finally: $E[l(x)] = \sum_{y \in T} E[C_{x,y}] = \underbrace{\frac{1}{m} + \frac{1}{m} + \cdots + \frac{1}{m}}_{n \text{ time}} = \frac{n}{m}$

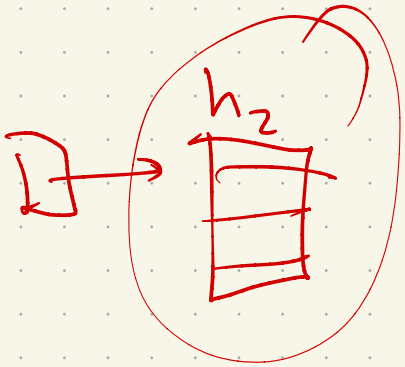
n things stored

Note: Practically, don't need a list:

↳ BST : $O(\log_2(\ell(x)))$

↳ vector/array

↳ another hash fcn




Also: Not saying $O(1)$ time expected in worst case!

That is $O(1 + \underbrace{\max_x \ell(x)}_{\text{not } O(1)})$

provably, more like $\log_2 n / \log_2 \log_2 n$

# Multiplicative hashing: Why all the fuss?

Some number theory:

given $a < p$, $a \neq 0$, there is only one $b < p$ s.t. $ab \bmod p \equiv 1$

Example: $p = 7$ :

| a | b |
|---|---|
| 1 | 1 |
| 2 | 4 |
| 3 | 5 |
| 4 | 2 |
| 5 | 3 |
| 6 | 6 |

| a | b |
|---|---|
| 1 | 1 |
| 2 | x |
| 3 | x |
| 4 | x |
| 5 | 5 |

In contrast: $n = 6$

Proof: Suppose $az \bmod p = az' \bmod p$

$z \neq z'$

$\Rightarrow a(z - z') \bmod p \equiv 0$

$\Rightarrow a(z - z') = k \cdot p$ for some $k$

but $a < p$.

$\Rightarrow$ $p$ can't be a factor of $a$

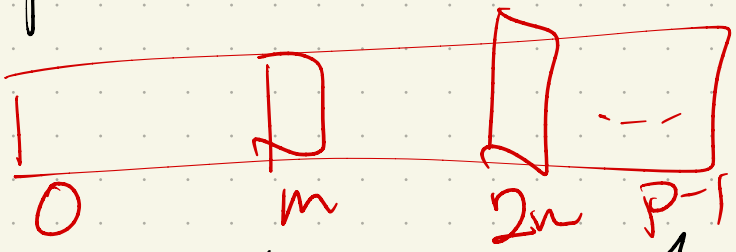
But both $z$ & $z' < p$ (and $> 0$);

$|z - z'| < p$, so can't be a factor.

Now, if we go back to $x, y \in U$, & use $h(x) = [ax \pmod p] \bmod m$

collisions can only happen if

$a(x-y) \bmod p$ is a multiple of $m$


How likely?

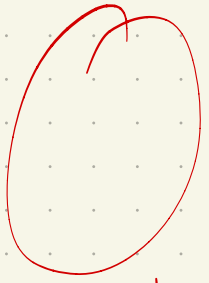
There are $\frac{p-1}{m}$ such values $\leq p-1$

So: collisions unlikely

# Binary multiplicative hashing:

Assume powers of 2: (restriction)

$$U = 2^w \qquad m = 2^\ell$$





Fix an odd $a \in \{1, \ldots, 2^w\}$

$$h_a(x) = \frac{ax \bmod 2^w}{2^{w-\ell}}$$

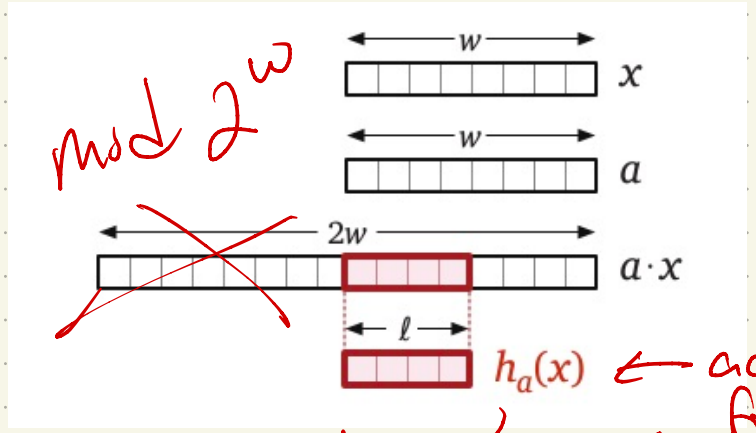
Example: $w = 32$, $\ell = 10$ (so table is $2^{10}$ = 1024)

$h(x) = a * x >> 22$

↳ bit shifting

Why?

- bit shifting → fast




- odd numbers are invertible mod $2^w$

  ⇒ multiplication permutes the numbers in the w bits

- Why high bits?

  Could use low, but then only depends on lower bits of w

Universal: Here,

$$\Pr_a[h_a(x) = h_a(y)] \leq \frac{1}{2^\ell} = \frac{1}{m}$$

$m = 2^\ell$

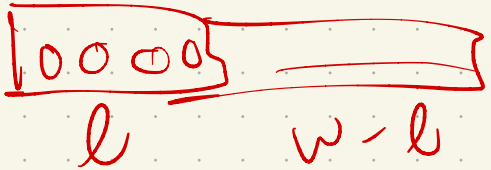
Take $x \neq y$, so $x - y \neq 0$.

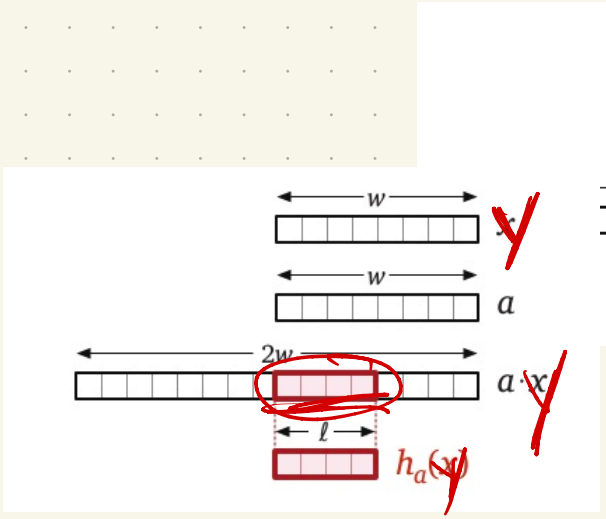
$$h_a(x) = h_a(y)$$

$$\iff \frac{ax \bmod 2^w}{2^{w-\ell}} = \frac{ay \bmod 2^w}{2^{w-\ell}}$$

top $\ell$ bits of

$a(x-y)$ are all 0



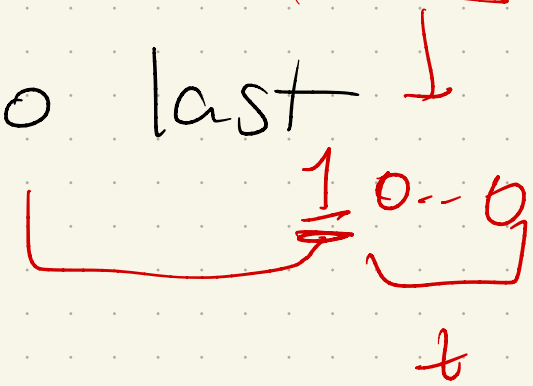
So: for random odd $a$, how likely is $a(x-y) \bmod 2^w$ to start with all 0's?

Some number theory:

factor out 2's: $x - y = 2^t \cdot d$ (odd)

$$a(x-y) = a \cdot 2^t \cdot d = 2^t (a \cdot d)$$

Both $a$ and $d$ are odd, so last $t$ bits: all 0



Essentially multiplying by random odd # is equivalent to permuting.

$$\Rightarrow \Pr[\text{top } \ell \text{ bits} = 0] = \frac{1}{2^\ell}$$

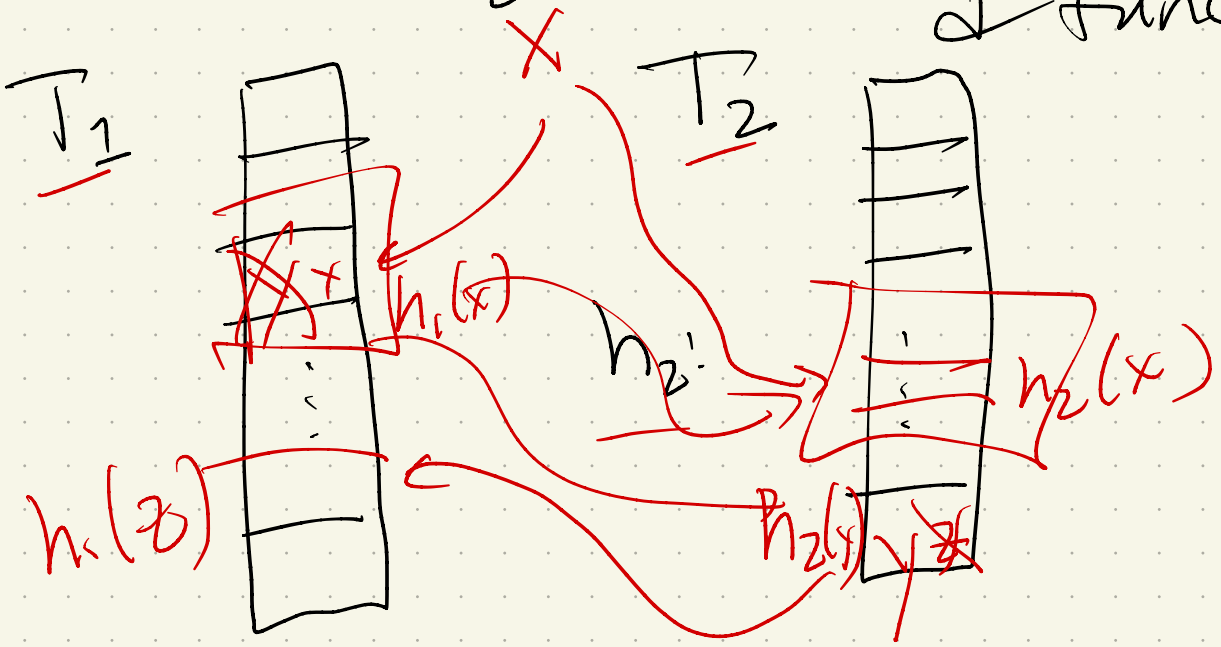
So equivalent to modular hashing:

- universal
- neither is k-uniform:
  consider $x, y, x+y$, then
  $a(x+y) = ax + ay$

But uses binary mixing for randomness
not algebraic field randomness
↳ fast.

# Cuckoo Hashing

Two hash tables & functions:




- Maintain: x is in $h_1(x)$ or $h_2(x)$

Search: $O(1)$

Insert: might have collision

Insert: Check $h_1(x)$:

If empty: insert

Otherwise: evict! $y$ in $h_1(x)$

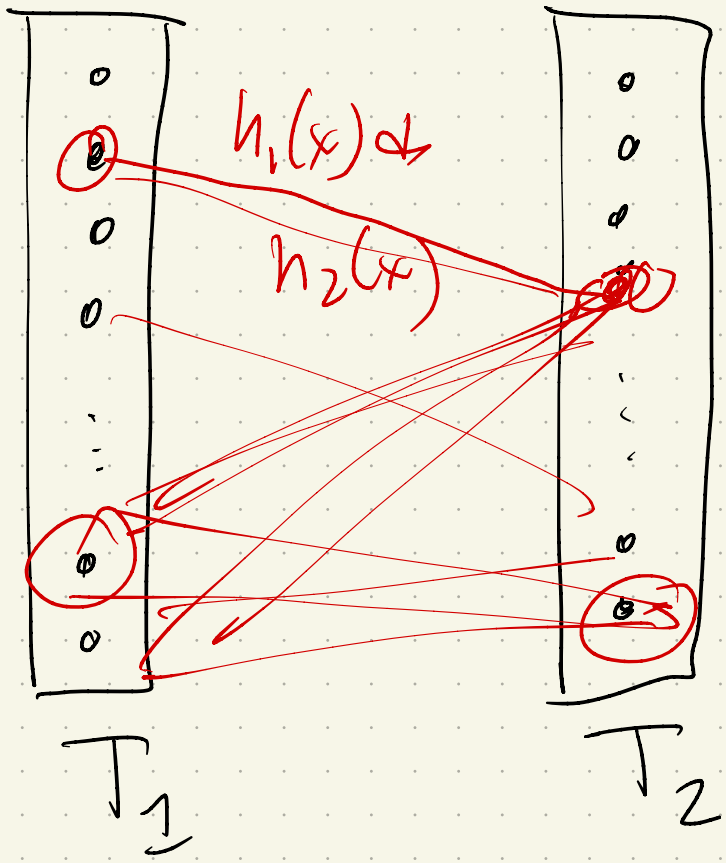
↳ go search $h_2(y)$

Cycle $x \to y \to z \cdots$

If runs too long:

stop & get new fcn

# Analysis: Cuckoo Graph




When are we stuck?

could have cycle

So: "random graph" ⇒ no cycles & short paths.

In practice:

# Cuckoo Hashing

Rasmus Pagh* and Flemming Friche Rodler

**BRICS****
Department of Computer Science
University of Aarhus, Denmark
{pagh,ffr}@brics.dk

in ESA, 2001

Why?

caching

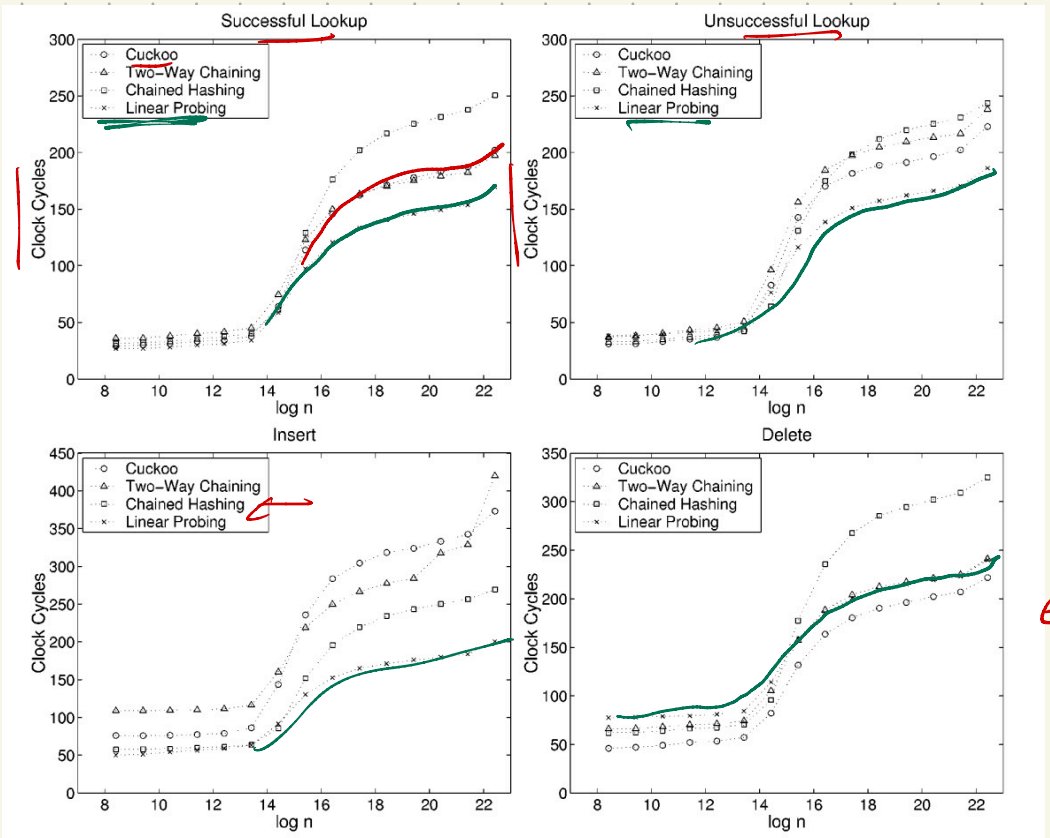



**Fig. 1.** The average time per operation in equilibrium for load factor 1/3.

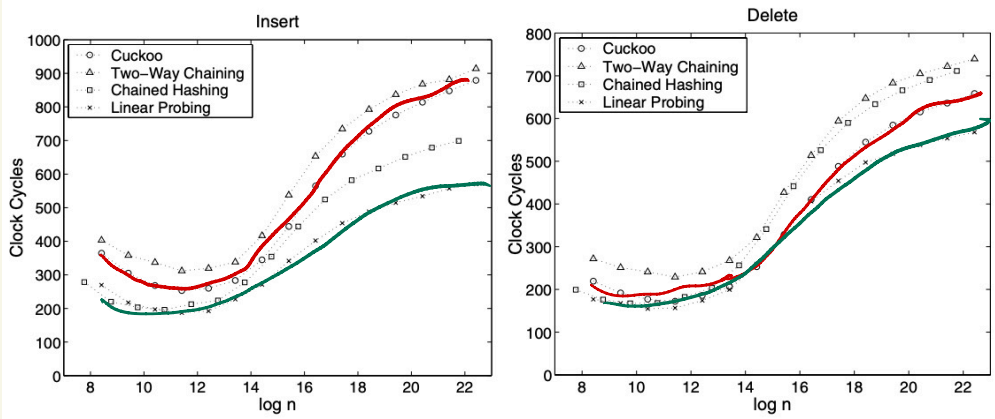



**Fig. 2.** The average time per insertion/deletion in a growing/shrinking dictionary for average load factor $\approx 1/3$.

# Recall: Linear probing

Linear probing: if we hash to a "full" spot, just walk down to open one.

h(x):

$\alpha = \frac{n}{m}$

$n = \frac{1}{2}m$

Insert

try h(x)+1

h(x)+2

⋮

→ empty spot C

is h(10) here?

Wost case: $O(n)$ insert or lookup

Expected: ?

If "uniform" "good" hashing, each cell is full with probability $\alpha = \frac{n}{m}$.

So, consider a run length $R$:

$R = 0$ if cell empty

$R = 1$ if $i$ full, $i+1$ empty

$R = 2$ if $i, i+1$ full, $i+2$ empty

etc.

$$\Pr[R > t] = \Pr[\text{cells } i \ldots i+t-1 \text{ full}]$$

$$= \frac{n}{m} \cdot \frac{n}{m} \cdots \frac{n}{m} = \alpha^t$$

$i \quad i+1 \quad \cdots \quad i+t-1$

$$E[R] = \sum_{t=1}^{\infty} Pr(R \geq t)$$

$$= \sum_{t=1}^{\infty} \alpha^t$$

what is this?

geometric series!

$\alpha < 1$

$n < m$

$$\Rightarrow E[R] = \frac{1}{1-\alpha}$$

# cells search if not present:

$$E[R+1] = \frac{1}{1-\alpha} - \alpha = \frac{\alpha}{1-\alpha}$$

constant depends on load parameter

In reality: "primary clusters" do happen. in practice longer runs than expected

But: This can help!

Why?

Cache loads

load nearby blocks

Caveat: Very active area!

NEWS | FREE ACCESS

# Speeding Up Hash Tables

Recent hash table development raises questions about an optimal solution.

Author: Sarah Underwood | Authors Info & Claims

Communications of the ACM, Volume 69, Issue 1 • Pages 11 - 13 • https://doi.org/10.1145/3772375

Published: 02 December 2025 Publication History

Check for updates

0 5,935 All formats PDF

Hash tables are one of the oldest and simplest data structures for storing elements and supporting deletions and queries. Invented in 1953, they underly most computational systems. Yet despite their ubiquity, or perhaps because of it, computer scientists continually strive to improve their performance with a view to achieving an optimal trade-off between time and space.

2022

# Iceberg Hashing: Optimizing Many Hash-Table Criteria at Once

Michael A. Bender* Alex Conway† Martín Farach-Colton ‡
William Kuszmaul § Guido Tagliavini ¶



**Abstract**

Despite being one of the oldest data structures in computer science, hash tables continue to be the focus of a great deal of both theoretical and empirical research. A central reason for this is that many of the fundamental properties that one desires from a hash table are difficult to achieve simultaneously; thus many variants offering different trade-offs have been proposed.

This paper introduces Iceberg hashing, a hash table that simultaneously offers the strongest known guarantees on a large number of core properties. Iceberg hashing supports constant-time operations while improving on the state of the art for space efficiency, cache efficiency, and low failure probability. Iceberg hashing is also the first hash table to support a load factor of up to $1 - o(1)$ while being stable, meaning that the position where an element is stored only ever changes when resizes occur. In fact, in the setting where keys are $\Theta(\log n)$ bits, the space guarantees that Iceberg hashing offers, namely that it uses at most $\log \binom{|U|}{n} + O(n \log \log n)$ bits to store $n$ items from a universe $U$, matches a lower bound by Demaine et al. that applies to any stable hash table.

Iceberg hashing introduces new general-purpose techniques for some of the most basic aspects of hash-table design. Notably, our indirection-free technique for dynamic resizing, which we call waterfall addressing, and our techniques for achieving stability and very-high probability guarantees, can be applied to any hash table that makes use of the front-yard/backyard paradigm for hash table design.