I & C SCI 46 Winter 2023
Lecture 10: Hash Tables II

How common are collisions?

$$\frac{\#\text{yes}}{\#\text{total}}$$

It's like the birthday paradox.
Imagine all 366 birthdays equally likely

- One person in a room
- Pick a person uniformly at random
  - They go into the room
  - They call "stop" if shared birthday

$$\frac{365}{366} \times \frac{364}{366} \times \frac{363}{366}$$

Side question: Average Class Size

- Some universities advertise average class size
  - 10 classes offered
  - One class has 910 enrolled
  - Other nine have 10 each

- Imagine every student takes only one class
- What is the average class size?

$1000/10 = 100$

- What do students think it is?

$\frac{1}{1000}(90 \cdot 10 + 910 \cdot 910) = 829$

Analysis of Hash Chaining

Recall $\alpha = n/m$ load factor

- **Worst-case** time for hashing with chaining?

- Let's consider *average-case* time.
  - Expected length of a bucket?

  - Time for an unsuccessful search?

Hash Functions

last time

For use in converting key to array index.

- Positive Integers: modular hashing
- Floating-point numbers
  - Multiply-and-round
  - Binary representation
- Strings
- Compound keys

$12.25 = 1100.01$

Requirements for hash functions

A good hash function for a given data type:

- Consistent
- Efficient to compute
- Uniformly distribute the set of keys

Ways to View Keys

Standard is one of two ways

- Each key is a tuple: $(x_1, x_2, \ldots, x_d)$

- Each key is a non-negative integer

Summing Components

Suppose your key is a d-tuple
$h(k) = \sum_{i=1}^{d} x_i$

$h(\text{"mike"}) = m + i + k + e$
$= 13 + 9 + 11 + 5 = 38$

$h(\text{"arts"}) = h(\text{"rats"}) = h(\text{"star"})$
$= h(\text{"tars"})$

Polynomial-Evaluation Functions

Choose non-zero $a \neq 1$ and:
$h(k) = x_1 a^{d-1} + x_2 a^{d-2} + \ldots + x_{d-1} a + x_d$

This can be written as:

$a = 37$

$h(k) = x_d + a(x_{d-1} + a(x_{d-2} + \ldots + a(x_3 + a(x_2 + a x_1)) \ldots))$

$h(\text{"mike"}) = m \cdot 37^3 + i \cdot 37^2 + k \cdot 37 + e$

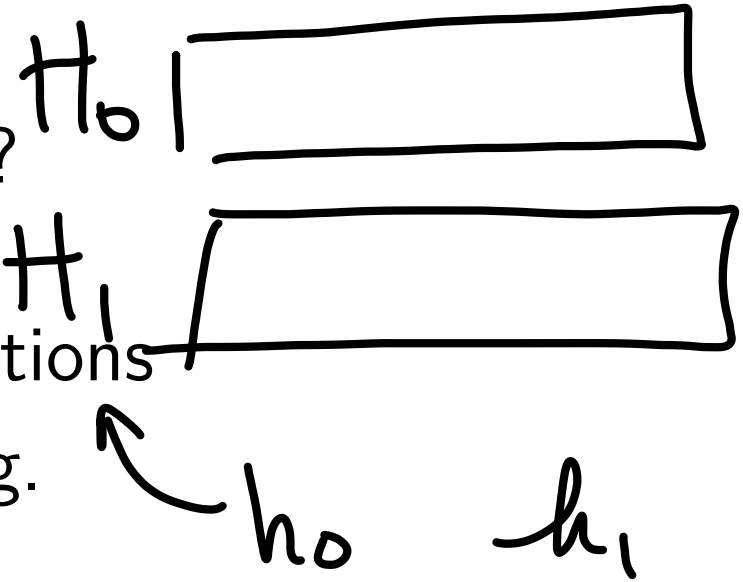
$= e + 37(k + 37(i + 37m))$

Approaches We've Seen So Far

Assuming suitable hash function

- Separate Chaining:
  - Insert: $\mathcal{O}(1)$ if LL reasonable
  - Search: expected $\alpha + 1$
- Linear and Quadratic Probing:
  - Can be $\mathcal{O}(n)$ insertion/search
- Slow down insert to improve search?

If one hash table is so great...



- Why not two hash tables?
  - Equal sizes
- Two (different) hash functions
- Direct storage, no probing.
- How do we search?

find x: $H_0[h_0(x)]$ or $H_1[h_1(x)]$

- How do we delete?

- How about insertion?

Example Insertion

Cuckoo Hashing

- $h_0(k) = k\%11$ $\qquad$ $h_1(k) = (k/11)\%11$

Insert: 12, 26, 92, 23, 28, 94, 15

0	1	2	3	4	5	6	7	8	9	10
	12			~~26~~ 92						

		26								

Visualizing Cuckoo Hashing

Key	12	26	92	23	28	94	15
h_0	1	4	4	1	6	6	4
h_1	1	2	8	2	2	8	1

0	1	2	3	4	5	6	7	8	9	10
	23			26		94				

	12	28						92		

0 1 2 3 4 5 6 7 8 9 10

0 1 2 3 4 5 6 7 8 9 10

Relevant Graph Theory

Theorem: Let G be an undirected connected graph. We can direct the edges of G such that no vertex has more than one incoming edge if and only if G has at most one cycle.

Relevant Graph Theory

Theorem: Let G be an undirected connected graph. We can direct the edges of G such that no vertex has more than one incoming edge if and only if G has at most one cycle.

Importance:

- Connected component has at most one cycle?
- What about when a second cycle?