Today: Hashing
- review:
  - dictionaries
  - chaining
  - simple uniform
  - universal hashing
    - why (useful)
    - how
  - perfect hashing
    - how
    - why (it works)

Dictionary problem: Abstract Data Type (ADT) maintain set of items, each with a key, subject to
  - insert(item): add item to set
  - delete(item): remove item from set
  - search(key): return item with key if it exists
- assume items have distinct keys (or that inserting new one clobbers old)
- easier than predecessor/successor problem solved by AVL/2-3 trees/skip lists & by van Emde Boas
Hashing [6.006]
- goal: \(O(1)\) time per operation \& \(O(n)\) space
- \(u = \#\) keys over all possible items
- \(n = \#\) keys/items currently in table
- \(m = \#\) slots in table
- hashing with chaining achieves \(\Theta(1 + x)\) time per op.
  \(\uparrow\) load factor \(w/m\)

Assuming simple uniform hashing:
\[
\Pr [\text{\(k_1 \neq k_2\)} \land h(k_1) = h(k_2)] \leq \frac{1}{m}
\]  
what you'd expect if totally uniform

- requires assuming input keys are random
- only works in average case
  (like Basic Quicksort)

We will remove this unreasonable assumption.

Etymology:
- English ‘hash’ (1650s) = cut into small pieces
- French ‘hacher’ = chop up
- Old French ‘hache’ (cf. English ‘hatchet’)
- Vulcan ‘la’ash’ = axe
Universal hashing:
- choose a random hash function \( h \) from \( \mathcal{H} \)
- require \( \mathcal{H} \) to be a universal hash family:
  \[
  \Pr\{ h(k) = h(k') \} \leq \frac{1}{m} \quad \text{for all } k \neq k'
  \]
- now just assuming \( h \) is random
- no assumption about input keys (like Randomized Quicksort)

**Theorem:** for \( n \) arbitrary distinct keys & for random \( h \in \mathcal{H} \), & \( \mathcal{H} \) universal
\[
E[\# \text{ keys colliding in a slot}] \leq 1 + \alpha \quad \Rightarrow \quad \frac{n}{m}
\]

**Proof:** consider keys \( k_1, k_2, \ldots, k_n \)
- let \( I_{i,j} = \begin{cases} 1 & \text{if } h(k_i) = h(k_j) \\ 0 & \text{else} \end{cases} \)

\[
E[\# \text{ keys hashing to same slot as } k_i] = E[\sum_{j=1}^{n} I_{i,j} ]
\]
\[
= \sum_{j=1}^{n} E[I_{i,j}] \quad \text{left linearity of expectation}
\]
\[
= \sum_{j=1}^{n} E[I_{i,j}] + E[I_{i,i}] = \Pr\{I_{i,j} = 1\} \quad \text{indicator random var.}
\]
\[
\leq \frac{1}{m} \quad \text{universality}
\]
\[
\leq \frac{n}{m} + 1
\]

\( \Rightarrow \) Insert, Delete, Search cost \( O(1 + \alpha) \) expected.
Do universal hash families exist? **YES:**

\[ \mathcal{H} = \{ \text{all hash functions} \} \quad h : \{0, 1, \ldots, u-1\} \rightarrow \{0, 1, \ldots, n-1\} \] is universal

... but this is useless:
- Storing \( h \) takes \( \lg(m) = \lg u \lg m \) bits \( \gg n \)
  - just like direct map table (big array)
- Would need to precompute \( u \) values
  \( \Rightarrow \Omega(u) \) time, possibly \( w(\# \text{operations}) \)

**Dot-product hash family:**
- Assume \( m \) is prime (find nearby prime)
- Assume \( u = m^r \) for integer \( r \) (round up else)
- View keys in base \( m \): \( k = (k_0, k_1, \ldots, k_{r-1}) \)
- For key \( a = (a_0, a_1, \ldots, a_{r-1}) \)
  - Define \( h_a(k) = \frac{1}{m} \sum_{i=0}^{r-1} a_i \cdot k_i \mod m \)
  - \( \dot{\prod} \)

\[ \mathcal{H} = \{ h_a \mid a \in \{0, 1, \ldots, u-1\} \} \]

- Storing \( h_a \in \mathcal{H} \) requires just storing 1 key, a
  - Word RAM model: manipulating \( O(1) \) machine words takes \( O(1) \) time,
  & "objects of interest" (here: keys)
  - fit in a machine word
  \( \Rightarrow \) computing \( h_a(k) \) takes \( O(1) \) time

\([O(\lg m u)] \) using just \(+ \&\cdot\) ~ can you do better?\]
Theorem: dot-product hash family $\mathcal{H}$ is universal

Proof: take any two keys $k \neq k'$

\[ \Rightarrow \text{differ in some digit, say } k_d \neq k'_d \]

- let $\not d = \{0, 1, \ldots, r-1\} \setminus \{d\}$

\[ \Pr_a \{ h_a(k) = h_a(k') \} \leq \frac{r-1}{r} \]

\[ \Pr_a \{ \sum_{i=0}^{r-1} a_i \cdot k_i = \sum_{i=0}^{r-1} a_i \cdot k'_i \mod m \} \leq \frac{r-1}{r} \]

\[ \Pr_a \{ \sum_{i \not \in d} a_i \cdot k_i + a_d \cdot k_d = \sum_{i \not \in d} a_i \cdot k'_i + a_d k'_d \mod m \} \leq \frac{r-1}{r} \]

\[ \Pr_a \{ \sum_{i \not \in d} a_i (k_i - k'_i) + a_d (k_d - k'_d) = 0 \mod m \} \leq \frac{r-1}{r} \]

\[ \Pr_a \{ a_d = -(k_d - k'_d)^{-1} \sum_{i \not \in d} a_i (k_i - k'_i) \mod m \} \leq \frac{r-1}{r} \]

\[ \text{because } a_d \text{ is independent from } a_{\not d} \]

\[ \Pr_a \{ a_{\not d} = \{a_{\not d} \} \} \leq \frac{r-1}{r} \]

\[ \Pr_a \{ a_d = f(k, k', a_{\not d}) \} \leq \frac{r-1}{r} \]

\[ \Pr_a \{ a_d = f(k, k', 0) \} \leq \frac{r-1}{r} \]

\[ \Pr_a \{ a_d = f(k, k', 1) \} \leq \frac{r-1}{r} \]

\[ \text{m prime } \Rightarrow \mathbb{Z}_m \text{ has multiplicative inverses} \]

\[ \sum_{a_{\not d}} \Pr_a \{ a_d = f(k, k', a_{\not d}) \} \leq \frac{r-1}{r} \]

\[ \sum_{a_{\not d}} \Pr_a \{ a_{\not d} = \{a_{\not d} \} \} \leq \frac{r-1}{r} \]

\[ = \sum_{a_{\not d}} \Pr_a \{ a_{\not d} = \{a_{\not d} \} \} \leq \frac{r-1}{r} \]

\[ = \sum_{a_{\not d}} \Pr_a \{ a_d = f(k, k', a_{\not d}) \} \leq \frac{r-1}{r} \]

\[ = \sum_{a_{\not d}} \Pr_a \{ a_{\not d} = \{a_{\not d} \} \} \leq \frac{r-1}{r} \]

\[ = \frac{1}{m} \]

\[ \frac{1}{m} \]

Another universal hash family: [CLRS]

- choose prime $p \geq u$ (once)

\[ - h_{ab}(k) = [(a \cdot k + b) \mod p] \mod m \]

\[ \Omega_a = \{ h_{ab} \mid a, b \in \{0, 1, \ldots, u-1\} \} \]

\[ \square \]
Static dictionary problem: given n keys to store in table, support Search(k)

⇒ no collisions

Perfect hashing: [Fredman, Komlós, Szemerédi 1984]
- polynomial build time w.h.p. (nearly linear)
- $O(1)$ time for Search, in worst case
- $O(n)$ space in worst case

Idea: 2-level hashing

1. pick $h_1: \{0,1,\ldots,n-1\} \rightarrow \{0,1,\ldots,m-1\}$ from a universal hash family
   for $m = \Theta(n)$ (e.g. nearby prime)
   - hash all items with chaining using $h_1$

2. for each slot $j \in \{0,1,\ldots,m-1\}$:
   - let $l_j = \#$ items in slot $j = |\{i \mid h(k_i) = j\}|$
   - pick $h_{2,j}: \{0,1,\ldots,n-1\} \rightarrow \{0,1,\ldots,m_j\}$ from a universal hash family
     for $l_j^2 \leq m_j \leq O(l_j^3)$ (e.g. nearby prime)
   - replace chain in 1 slot $j$ with hashing-with-chaining using $h_{2,j}$

Space = $O(n + \sum_{j=0}^{m-1} l_j^2)$
- to guarantee space = $O(n)$:

1.5 if $\sum_{j=0}^{m-1} l_j^2 > c n$ then redo step 1
Search time = $O(1)$ for first table ($h_1$) 
+ $O(\max \text{ chain size in second table})$
- to guarantee = $O(1)$:

2.5: while $h_{2,j}(k_i) = h_{2,j}(k_{i'})$ for any $i \neq i'$, repick $h_{2,j}$ & rehash those $l_j$ items

$\Rightarrow$ no collisions at second level!

**Build time:** 1 & 2 are $O(n)$. 1.5 & 2.5?

2.5: $\Pr \bigg\{ h_{2,j}(k_i) = h_{2,j}(k_{i'}) \text{ for some } i \neq i' \bigg\}$

\[ \leq \sum_{i \neq i'} \Pr \bigg\{ h_{2,j}(k_i) = h_{2,j}(k_{i'}) \bigg\} \] 
\[ \leq \left( \frac{l_j}{2} \right) \cdot \frac{1}{l_j^2} \] 
\[ < \frac{1}{2} \] 
(Birthday Paradox)

$\Rightarrow$ each trial is like a coin flip, tails $\Rightarrow$ OK
$\Rightarrow$ $E[\# \text{ trials}] \leq 2$
$\&$ #$\text{ trials} = O(\log n)$ w.h.p. (by Lecture 7)

- Chernoff bound $\Rightarrow l_j = O(\log n)$ w.h.p.
$\Rightarrow$ each trial $O(\log n)$ time (also obviously $O(n)$)
- must do this for each $j$
$\Rightarrow O(n \log^2 n)$ time w.h.p. (or obviously $O(n^2 \log n)$)
\( \mathbb{E} \left[ \sum_{j=0}^{m-1} l_j^2 \right] = \mathbb{E} \left[ \sum_{i=1}^{n} \sum_{i'=1}^{n} \text{I}_{i \neq i'} \right] \)

indicator rand. var. = \( \begin{cases} 1 \text{ if } h_1(k_i) = h_1(k_{i'}) \\ 0 \text{ else} \end{cases} \)

\[
= \sum_{i=1}^{n} \sum_{i'=1}^{n} \mathbb{E} [\text{I}_{i \neq i'}] \leq \text{linearity of expectation} \\
= \sum_{i=1}^{n} \mathbb{E} [\text{I}_{i \neq i'}] + 2 \sum_{i \neq i'} \mathbb{E} [\text{I}_{i \neq i'}] \\
\leq n + 2 \binom{n}{2} \cdot \frac{1/m}{n} \leq \Theta(n) \quad \text{because } m = \Theta(n) \\
\text{Pr} \left\{ \sum_{j=0}^{m-1} l_j^2 \geq c \cdot n \right\} \leq \frac{\mathbb{E} \left[ \sum_{j=0}^{m-1} l_j^2 \right]}{c \cdot n} \leq \frac{1}{2} \quad \text{for suff. large const. } c \\
\Rightarrow \mathbb{E} [\text{# trials}] \leq 2 \\
\text{& } \text{# trials } = O(\lg n) \text{ w.h.p.} \\
\Rightarrow (i) \text{ & (i.5) take } O(n \lg n) \text{ w.h.p.}