Hashing: (Unordered) dictionary
- stores key-value pairs in array table [0..m-1]
- supports basic dict. ops. (insert, delete, find) in O(1) expected time
- does not support ordered ops (getMin, findUp, ...)
- simple, practical, widely used

Recap: So far, ordered dicts.
- insert, delete, find
- Comparison-based: <, ==, >
- getMin, getMax, getK, findUp...
- Query/Update time: O(log n)

Universal Hashing:
- Even better — randomize!
- Let H be a family of hash fns
- Select h \in H randomly
- If x \neq y, then \text{Prob}(h(x) = h(y)) = \frac{1}{m}

\text{E.g. Let } p \text{ - large prime, } a \in [1..p-1]
\text{b} \in [0..p-1] \text{ all random}

\begin{align*}
    h_{a,b}(x) &= ((ax+b) \mod p) \mod m
\end{align*}

Overview:
- To store n keys, our table should (ideally) be a bit larger (e.g., m \geq c \cdot n, c=1.25)
- Load factor:
  \begin{align*}
      \lambda &= \frac{n}{m}
  \end{align*}
- Running times increase as \lambda \to 1
- Hash function:
  \begin{align*}
      h &: \text{Keys} \to [0..m-1] \\
      \text{Should scatter keys random.} \\
      \text{Need to handle collisions.} \\
  \end{align*}

Load factor:
- \begin{align*}
      \lambda &= \frac{n}{m} \\
      m &= 1 \quad h(x) \\
      m &= 1 \\
    \end{align*}
- \text{Efficient to compute}
- \text{Produce few collisions}
- \text{Use every bit in key}
- \text{Break up natural clusters}

Good Hash Function:
- \text{E.g. Java variable names: temp1, temp2, temp3}
- \text{Collision resolution}

Common Examples:
- Division hash:
  \begin{align*}
      h(x) &= x \mod m
  \end{align*}
- Multiplicative hash:
  \begin{align*}
      h(x) &= (ax \mod p) \mod m \\
      a, p - \text{large prime numbers}
  \end{align*}
- Linear hash:
  \begin{align*}
      h(x) &= (ax + b) \mod p \mod m \\
      a, b, p - \text{large primes}
  \end{align*}
Overview:
- Separate Chaining
- Open Addressing:
  - Linear probing
  - Quadratic probing
  - Double hashing

Collision Resolution:
If there were no collisions, hashing would be trivial!

- Linear probing
  \[ \text{insert}(x,v) \rightarrow \text{table}[h(x)] = (x,v) \]
- Quadratic probing
  \[ \text{find}(x) \rightarrow \text{return table}[h(x)] \]
- Double hashing
  \[ \text{delete}(x) \rightarrow \text{table}[h(x)] = \text{null} \]

If \( \lambda < \lambda_{\text{min}} \) or \( \lambda > \lambda_{\text{max}} \)? Rehash!
- Alloc. new table size = \( \frac{n}{\lambda_0} \)
- Compute new hash function \( h \)
- Copy each \( x,v \) from old to new using \( h \)
- Delete old table

Separate Chaining:
- \( \text{table}[i] \) is head of linked list of keys that hash to \( i \).

Example:

<table>
<thead>
<tr>
<th>Keys (x)</th>
<th>( h(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Table:

\[ \lambda = \frac{6}{8} \]

Analysis:
- Recall load factor \( \lambda = \frac{n}{m} \)
- \( n \) = # of keys
- \( m \) = table size

Expected search time:
- If \( x \) found (successful):
  \[ E_{\text{sc}} = 1 + \frac{\lambda}{2} \]
- If \( x \) not found (unsuccessful):
  \[ E_{\text{sc}} = | + \lambda | \]

Example:
- \( n = 6 \)
- \( m = 8 \)

Proof:
- On avg., each list has \( \frac{n}{m} = \lambda \)
  - Success: 1 for head + half the list
  - Unsuccess: 1 + all the list

Thm:
- Amortized time for rehashing is
  \[ 1 + \left(\frac{2\lambda_{\text{max}}}{\lambda_{\text{max}} - \lambda_{\text{min}}}\right) \]

How to control \( \lambda \)?
- Rehashing: If table too dense / too sparse, reallocate to new table of ideal size
- Designer: \( \lambda_{\text{min}}, \lambda_{\text{max}} \) - allowed \( \lambda \) value
  \[ \lambda_0 = \frac{\lambda_{\text{min}} + \lambda_{\text{max}}}{2} \]

If \( \lambda < \lambda_{\text{min}} \) or \( \lambda > \lambda_{\text{max}} \) ...
Open Addressing:
- Special entry ("empty") means this slot is unoccupied.
- Assume $\lambda \leq 1$
- To insert key $x$:
  - Check $h(x)$ if not empty try $h(x)$, $h(x)+1$, $h(x)+2$, ...

Collision Resolution (cont.):
- Separate chaining is efficient, but uses extra space (nodes, pointers,...)
- Can we just use the table itself?

Analysis: Improves secondary clustering
- May fail to find empty entry
  - Try $m=4$, $j^2 \mod 4 = 0, 1$ but $mod 10, 1, 3, 4, 5$ not $2, 3$
- How bad is it? It will succeed if $\lambda < \frac{1}{2}$.
- Quadratic Residue $j \in \mathbb{N} | j^2 \mod m | = \frac{m}{2}$

Thm: If quad. probing used + m to prime, the the first $\lfloor m/2 \rfloor$ probe locations are distinct.

Pf: See latex notes.

Analysis: Improves secondary clustering
- Can we just use the table itself?

Hashing III

Probe sequence
- What's the best probe sequence?

Linear Probing:
- $h(x)$, $h(x)+1$, $h(x)+2$, ...

Simple, but is it good?
- $x$: $d, e, p, n, t$
- $h(x)$: $0, 2, 3, 0, 1$
- $\lambda$: $\frac{1}{2}$ - distinctive - nonetheless - collision

Table:

- $h(x)$:
- $0, 2, 3, 0, 1$

Clustering
- Clusters form when keys are hashed to nearby locations
- Spread them out?

Quadratic Probing:
- $h(x)$, $h(x)+1$, $h(x)+4$, $h(x)+9$, ...

Thm: $S_{LP} = \frac{1}{2} \left( 1 + \frac{1}{1-\lambda} \right)$

$U_{LP} = \frac{1}{2} \left( 1 + \frac{1}{1-\lambda} \right)^2$

Obs: As $\lambda \to 1$ times increase rapidly.

$\lambda = 0.90$
Double Hashing:
(Best of the open-addressing methods)
- Probe sequence det'd by second hash fn.: \( g(x) \)
  \[ h(x) + \{0, g(x), 2g(x), 3g(x), \ldots \} \mod m \]

Recall:
Separate Chaining:
Fastest but uses extra space (linked list)
Open Addressing:
Linear probing: clustering
Quadratic probing:

Why does bust up clusters?
Even if \( h(x) = h(y) \) [collision], it is very unlikely that \( g(x) = g(y) \)
⇒ Probe sequences are entirely different!
\( g(x) \) - relatively prime to \( m \) (no common factors)

Analysis:
Defs:
\( S_{DH} \) = Expected search time of doub. hash. if successful
\( U_{DH} \) = Exp. if unsuccessful
Recall: Load factor \( \lambda = n/m \)

\( S_{DH} = \frac{1}{\lambda} \ln \left( \frac{1}{1-\lambda} \right) \)
\( U_{DH} = \frac{1}{1-\lambda} \)

\( \lambda \): 0.5 0.75 0.95 0.99
\( U_{DH} \): 2 4 20 100
\( S_{DH} \): 1.39 1.89 3.15 4.65

Proof is nontrivial (skip)

Delete(\( x \)): Apply find(\( x \))
→ Not found ⇒ error
→ Found ⇒ set to "empty"

Problem:
insert(\( a \)): \( h(\lambda) \)
delete(\( a \)): \( h(\lambda) \)
find(\( a \)): \( h(a) \)

Find(\( x \)): Visit entries on probe sequence until:
- found \( x \) ⇒ return \( v \)
- hit empty ⇒ return null
find(\( x \)): \( h(x) \)
→ Not found!

Dictionary Operations:
Insert(\( x,v \)): Apply probe sequence until finding first empty slot.
- Insert(\( x,v \)) here.
  (If \( x \) found along the way ⇒ duplicate key error!)
Announcements: 11/10

Doom & Gloom
- Prog Assign 2 - Due today 11/5
- HW 3 - almost done
  Still - Point values
  - Challenge problem
  Due Tue, 11/15 - start of class
  - no late
  - turn in whatever you done

- Midterm 2 - Thu, Nov 17
  - 2 cheat sheets

#null ptr cells
intersect rect
is \(O(\sqrt{n} + k)\)

\[ k = \#\text{pts inside} \]