Learning Objectives

Review what a hash table is and what its key weakness is

Introduce closed hashing strategies
A Hash Table based Dictionary

```python
1  d = {}
2  d[k] = v
3  print(d[k])
```

A **Hash Table** consists of three things:

1. A hash function

2. A data storage structure

3. A method of addressing *hash collisions*
Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

- **Open Hashing**: store $k,v$ pairs externally
- **Closed Hashing**: store $k,v$ pairs in the hash table
Open Hashing: Separate Chaining

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>B+</td>
<td>2</td>
</tr>
<tr>
<td>Anna</td>
<td>A-</td>
<td>4</td>
</tr>
<tr>
<td>Alice</td>
<td>A+</td>
<td>4</td>
</tr>
<tr>
<td>Betty</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>Brett</td>
<td>A-</td>
<td>2</td>
</tr>
<tr>
<td>Greg</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>Sue</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>Ali</td>
<td>B+</td>
<td>4</td>
</tr>
<tr>
<td>Laura</td>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>Lily</td>
<td>B+</td>
<td>7</td>
</tr>
</tbody>
</table>
Simple Uniform Hashing Assumption

Given table of size $m$, a simple uniform hash, $h$, implies

$$\forall k_1, k_2 \in U \text{ where } k_1 \neq k_2, \ Pr(h[k_1] = h[k_2]) = \frac{1}{m}$$

**Uniform:** keys are equally likely to hash to any position

**Independent:** key hash values are independent of other keys
Separate Chaining Under SUHA

Under SUHA, a hash table of size $m$ and $n$ elements:

find runs in: __________.

insert runs in: __________.

remove runs in: __________.
Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

- **Open Hashing**: store $k,v$ pairs externally
- **Closed Hashing**: store $k,v$ pairs in the hash table
Collision Handling: Probe-based Hashing

$S = \{ 1, 8, 15 \}$

$h(k) = k \% 7$

$|S| = n$

$\mid \text{Array} \mid = m$
Collision Handling: Linear Probing

\( S = \{ 16, 8, 4, 13, 29, 11, 22 \} \) \hspace{1cm} |S| = n

\( h(k) = k \mod 7 \) \hspace{1cm} |Array| = m

\( h(k, i) = (k + i) \mod 7 \)

Try \( h(k) = (k + 0) \mod 7 \), if full...
Try \( h(k) = (k + 1) \mod 7 \), if full...
Try \( h(k) = (k + 2) \mod 7 \), if full...
Try ...
Collision Handling: Linear Probing

\[ S = \{ 16, 8, 4, 13, 29, 11, 22 \} \quad \text{\(|S| = n\)} \]
\[ h(k, i) = (k + i) \mod 7 \quad \text{\(|\text{Array}| = m\)} \]

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
</tr>
</tbody>
</table>

\_find(29)
Collision Handling: Linear Probing

$S = \{ 16, 8, 4, 13, 29, 11, 22 \}$  
$|S| = n$

Array = m

$$h(k, i) = (k + i) \% 7$$

$\_remove(16)$
A Problem w/ Linear Probing

Primary clustering:

```
0
1 1
2 1'
3 3
4 1"
5 3'
6
7
8
9
```

Description:

Remedy:
Collision Handling: Quadratic Probing

\[ S = \{ 16, 8, 4, 13, 29, 12, 22 \} \]
\[ |S| = n \]
\[ |Array| = m \]

\[ h(k) = k \mod 7 \]

Try \[ h(k) = (k + 0) \mod 7 \], if full...
Try \[ h(k) = (k + 1 \times 1) \mod 7 \], if full...
Try \[ h(k) = (k + 2 \times 2) \mod 7 \], if full...
Try ...

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
Collision Handling: Quadratic Probing

$S = \{ 16, 8, 4, 13, 29, 12, 22 \}$

$h(k) = k \% 7$

$|S| = n$

$|\text{Array}| = m$

_find(11)

_remove(16)
A Problem w/ Quadratic Probing

Secondary clustering:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0'</th>
<th>0''</th>
<th>0'''</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0''</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>0'''</td>
<td></td>
</tr>
</tbody>
</table>

Description:

Remedy:
Collision Handling: Double Hashing

$S = \{16, 8, 4, 13, 29, 11, 22\}$  \hspace{1cm} |S| = n

$h_1(k) = k \% 7$ \hspace{1cm} |Array| = m

$h_2(k) = 5 - (k \% 5)$

$$h(k, i) = (h_1(k) + i*h_2(k)) \% 7$$

Try $h(k) = (k + 0*h_2(k)) \% 7$, if full...

Try $h(k) = (k + 1*h_2(k)) \% 7$, if full...

Try $h(k) = (k + 2*h_2(k)) \% 7$, if full...

Try ...
Running Times
(Expectation under SUHA)

Open Hashing:
insert: __________.
find/ remove: __________.

Closed Hashing:
insert: __________.
find/ remove: __________.

(Don’t memorize these equations, no need.)
Running Times

(Don’t memorize these equations, no need.)

The expected number of probes for find(key) under SUHA

Linear Probing:
• Successful: $\frac{1}{2}(1 + \frac{1}{1-\alpha})$
• Unsuccessful: $\frac{1}{2}(1 + \frac{1}{1-\alpha})^2$

Double Hashing:
• Successful: $\frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)$
• Unsuccessful: $\frac{1}{1-\alpha}$

Separate Chaining:
• Successful: $1 + \frac{\alpha}{2}$
• Unsuccessful: $1 + \alpha$

Instead, observe:
- As $\alpha$ increases:
- If $\alpha$ is constant:
Running Times

The expected number of probes for find(key) under SUHA

Linear Probing:
• Successful: $\frac{1}{2} \left(1 + \frac{1}{1-\alpha}\right)$
• Unsuccessful: $\frac{1}{2} \left(1 + \frac{1}{1-\alpha}\right)^2$

Double Hashing:
• Successful: $\frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)$
• Unsuccessful: $\frac{1}{1-\alpha}$
Resizing a hash table

How do we resize?
<table>
<thead>
<tr>
<th></th>
<th>Hash Table</th>
<th>Array List</th>
<th>Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Find</strong></td>
<td>Amortized:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Worst Case:</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Insert (Order Agnostic)</strong></td>
<td>Amortized:</td>
<td>Amortized:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Worst Case:</td>
<td>Worst Case:</td>
<td></td>
</tr>
<tr>
<td><strong>Remove (By Value)</strong></td>
<td>Amortized:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Worst Case:</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Storage Space</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
On Wednesday: More uses for hash functions!
Choosing a Hash Function

Python has a built-in hash! It’s pretty good if you run everything at once.

```python
print(hash("I can pass in any string!"))
print(hash(205811))
```
Choosing a Hash Function

If you want something that is persistently deterministic, find a seeded hash

```python
import mmh3
print(mmh3.hash("I can pass in any string!", 10))  # I got: -565691678
print(mmh3.hash("I can pass in any string!", 50))  # I got: -947521776
print(mmh3.hash("I can pass in any string!", 12))  # I got: 1680496801
```
Bonus Slides
Hash Function (Division Method)

Hash of form: \( h(k) = k \% m \)

Pro:

Con:
Hash Function (Multiplication Method)

Hash of form: \( h(k) = \lfloor m(kA \mod 1) \rfloor, \ 0 \leq A \leq 1 \)

Pro:

Con:
Hash Function (Universal Hash Family)

Hash of form: $h_{ab}(k) = ((ak + b) \mod p) \mod m$, $a, b \in \mathbb{Z}_p^*$, $\mathbb{Z}_p$

$\forall k_1 \neq k_2, \ Pr_{a,b}(h_{ab}[k_1] = h_{ab}[k_2]) \leq \frac{1}{m}$

Pro:

Con: