Hashing!
(At long last)
So you want to write a set...

And you want to ask “Is DC in this set?”
So you want to write a set...

AK | NM...thousands of elements...

NC | CA

And you want to ask “Is DC in this set?”
HashCode: “I’m near index $k$”

At most a constant distance from index $k$, in fact.
HashCode: “I’m near index $k$”

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238
HashCode: “I’m near index $k$”

```
AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238
```

```
table[hashCode % table.length] = v;
```
HashCode: “I’m near index $k$”

```
AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238
```

```
table[hashCode % table.length] = v;
```
**HashCode:** “I’m near index $k$”

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

AK: 17  
NC: 31  
NY: 729  
TX: 312425  
WA: 53623238

```java
int table[hashCode % table.length] = v;
```
HashCode: “I’m near index $k$”

<table>
<thead>
<tr>
<th></th>
<th>NC</th>
<th></th>
<th></th>
<th></th>
<th>AK</th>
<th></th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

AK: 17  
NC: 31  
NY: 729  
TX: 312425  
WA: 53623238

table[hashCode % table.length] = v;
HashCode: “I’m near index $k$”

<table>
<thead>
<tr>
<th></th>
<th>NC</th>
<th></th>
<th></th>
<th>TX</th>
<th></th>
<th>AK</th>
<th></th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

```
    table[hashCode % table.length] = v;
```
**HashCode:** “I’m near index $k$”

<table>
<thead>
<tr>
<th></th>
<th>NC</th>
<th></th>
<th></th>
<th></th>
<th>TX</th>
<th></th>
<th>AK</th>
<th>WA</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

AK: 17  
NC: 31  
NY: 729  
TX: 312425  
WA: 53623238

```java
table[hashCode % table.length] = v;
```
HashCode: “I’m near index $k$”

<table>
<thead>
<tr>
<th></th>
<th>NC</th>
<th></th>
<th></th>
<th>TX</th>
<th></th>
<th>AK</th>
<th>WA</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

What’s the Big O of lookup?

```java
table[hashCode % table.length] = v;
```
**HashCode:** “I’m near index $k$”

<table>
<thead>
<tr>
<th></th>
<th>NC</th>
<th></th>
<th>TX</th>
<th></th>
<th>AK</th>
<th>WA</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>AK: 17</th>
<th>OH: 76</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC: 31</td>
<td>OR: 2</td>
<td></td>
</tr>
<tr>
<td>NY: 729</td>
<td>FL: 43803</td>
<td></td>
</tr>
<tr>
<td>TX: 312425</td>
<td>SC: 9000</td>
<td></td>
</tr>
<tr>
<td>WA: 53623238</td>
<td>DC: 75</td>
<td></td>
</tr>
</tbody>
</table>

```java
table[hashCode % table.length] = v;
```
HashCode: “I’m near index k”

table[hashCode % table.length] = v;
HashCode: “I’m near index $k$”

<table>
<thead>
<tr>
<th></th>
<th>NC</th>
<th>OR</th>
<th>TX</th>
<th>OH</th>
<th>AK</th>
<th>WA</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75

```
table[hashCode % table.length] = v;
```
**HashCode:** “I’m near index $k$”

<table>
<thead>
<tr>
<th>State</th>
<th>AK</th>
<th>NC</th>
<th>OR</th>
<th>FL</th>
<th>TX</th>
<th>OH</th>
<th>AK</th>
<th>WA</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AK:</td>
<td>17</td>
</tr>
<tr>
<td>NC:</td>
<td>31</td>
</tr>
<tr>
<td>NY:</td>
<td>729</td>
</tr>
<tr>
<td>TX:</td>
<td>312425</td>
</tr>
<tr>
<td>WA:</td>
<td>53623238</td>
</tr>
</tbody>
</table>

```
table[hashCode % table.length] = v;
```
**HashCode:** “I’m near index $k$”

<table>
<thead>
<tr>
<th>SC</th>
<th>NC</th>
<th>OR</th>
<th>FL</th>
<th>TX</th>
<th>OH</th>
<th>AK</th>
<th>WA</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

- AK: 17
- NC: 31
- NY: 729
- TX: 312425
- WA: 53623238
- OH: 76
- OR: 2
- FL: 43803
- SC: 9000
- DC: 75

```java
table[hashCode % table.length] = v;
```
HashCode: “I’m near index $k$”

<table>
<thead>
<tr>
<th>SC</th>
<th>NC</th>
<th>OR</th>
<th>FL</th>
<th>TX</th>
<th>OH</th>
<th>AK</th>
<th>WA</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75

```java
table[hashCode % table.length] = v;
```
HashCode: “I’m near index $k$”

table[hashCode % table.length] = v;
HashCode: “I’m near index k”

Draw the resulting table.

```java
table[hashCode % table.length] = v;
```
HashCode: “I’m near index $k$”

<table>
<thead>
<tr>
<th>SC</th>
<th>NC</th>
<th>OR</th>
<th>FL</th>
<th>AL</th>
<th>TX</th>
<th>OH</th>
<th>AK</th>
<th>WA</th>
<th>NY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 → AZ → AR

What’s the Big O of lookup?

<table>
<thead>
<tr>
<th>AK</th>
<th>NC</th>
<th>NY</th>
<th>TX</th>
<th>WA</th>
<th>OH</th>
<th>OR</th>
<th>FL</th>
<th>SC</th>
<th>DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>31</td>
<td>729</td>
<td>312425</td>
<td>53623238</td>
<td>76</td>
<td>2</td>
<td>43803</td>
<td>9000</td>
<td>75</td>
</tr>
</tbody>
</table>

Monday, November 26, 12
What do we do about collisions?
Pausing for a moment.

http://goo.gl/qB9DP
After adding $n$ elements to the table, we have $\approx n/k$ elements in each bucket.

Aside: this can be formalized. If we assume that our hash function is drawing from the buckets uniformly at random, we will have $n/k$ elements per bucket in expectation. As you know, actually using random won’t work, so this is a slightly optimistic analysis.
After adding $n$ elements to the table, we have $\approx n/k$ elements in each bucket.

Aside: this can be formalized. If we assume that our hash function is drawing from the buckets uniformly at random, we will have $n/k$ elements per bucket in expectation. As you know, actually using random won’t work, so this is a slightly optimistic analysis.
Rehashing

$k = 3$  

$\begin{align*}
\text{SC} & \quad \text{NC} & \quad \text{NY} \\
L &= 1.0
\end{align*}$

- SC: 9
- NC: 37
- NY: 98
- TX: 312425
- WA: 53623238
- OH: 76
- OR: 2
- FL: 43803
- SC: 9000
- DC: 75
- AL: 44
- AZ: 0
- AR: 90
- CA: 105
- CO: 205
- CT: 305
- DE: 405
Rehashing

$k = 3$

\[ L = \frac{4}{3} \]
Rehashing

$k = 3$  

\[
\begin{align*}
SC & \quad NC & \quad NY \\
\downarrow & & \\
TX & & \\
\downarrow & & \\
WA & & \\
\end{align*}
\]

$L = 5/3$

SC: 9  
NC: 37  
NY: 98  
TX: 312425  
WA: 53623238  
OH: 76  
OR: 2  
FL: 43803  
SC: 9000  
DC: 75  
AL: 44  
AZ: 0  
AR: 90  
CA: 105  
CO: 205  
CT: 305  
DE: 405
Rehashing

$k = 3$

SC \rightarrow NC \rightarrow NY

OH \rightarrow TX

WA

$L = 2$

SC: 9
NC: 37
NY: 98
TX: 3,124,25
WA: 536,232,38
OH: 76
OR: 2
FL: 43,803
SC: 9,000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405
A thoughtful person would ponder why I’m using prime-number-sized tables, and not powers of two.
A thoughtful person would ponder why I’m using prime-number-sized tables, and not powers of two.
Rehashing

A thoughtful person would ponder why I’m using prime-number-sized tables, and not powers of two.
A thoughtful person would ponder why I’m using prime-number-sized tables, and not powers of two.
A thoughtful person would ponder why I’m using prime-number-sized tables, and not powers of two.
A thoughtful person would ponder why I’m using prime-number-sized tables, and not powers of two.
Rehashing

A thoughtful person would ponder why I’m using prime-number-sized tables, and not powers of two.
Load Factor

http://goo.gl/fI7wt
Not just chains

\[ k = 3 \]
Not just chains

$k = 3$

SC
Not just chains

$k = 3$

SC | VT

SC: 9
VT: 9000
DC: 99

Monday, November 26, 12
Not just chains: *probing*

\[ k = 3 \]

\[
\begin{array}{ccc}
SC & VT & DC \\
\end{array}
\]

SC: 9
VT: 9000
DC: 99