Alternative: Use Empty Space in the Table

Insert:
38
19
8
109
10

Try $h(K)$.
If full, try $h(K)+1$.
If full, try $h(K)+2$.
If full, try $h(K)+3$.
Etc...
Alternative: Use Empty Space in the Table

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>38</td>
<td></td>
<td>38</td>
<td></td>
</tr>
</tbody>
</table>

Try $h(K)$.
If full, try $h(K)+1$.
If full, try $h(K)+2$.
If full, try $h(K)+3$.

Etc...

Insert:
- 38
- 19
- 8
- 109
- 10
Alternative: Use Empty Space in the Table

Insert:
38
19
8
109
10

Try \( h(K) \).
If full, try \( h(K) + 1 \).
If full, try \( h(K) + 2 \).
If full, try \( h(K) + 3 \).
Etc...
Alternative: Use Empty Space in the Table

<table>
<thead>
<tr>
<th>Insert:</th>
<th>38</th>
<th>19</th>
<th>8</th>
<th>109</th>
<th>10</th>
</tr>
</thead>
</table>

Try \( h(K) \).
If full, try \( h(K)+1 \).
If full, try \( h(K)+2 \).
If full, try \( h(K)+3 \).
Etc...

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>38</td>
</tr>
<tr>
<td>9</td>
<td>19</td>
</tr>
</tbody>
</table>
Alternative: Use Empty Space in the Table

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>109</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></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>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Insert:
- 38
- 19
- 8
- 109
- 10

Try \( h(K) \).
If full, try \( h(K) + 1 \).
If full, try \( h(K) + 2 \).
If full, try \( h(K) + 3 \).
Etc...
**Alternative: Use Empty Space in the Table**

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>109</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>38</td>
</tr>
<tr>
<td>9</td>
<td>19</td>
</tr>
</tbody>
</table>

**Insert:**

```
38
19
8
109
10
```

Try \( h(K) \).

If full, try \( h(K) + 1 \).

If full, try \( h(K) + 2 \).

If full, try \( h(K) + 3 \).

Etc...
Open Addressing

This approach is an example of **open addressing**: 
After a collision, try “next” spot. 
If there’s another collision, try another, etc.

Finding the next available spot is called **probing**:

0\(^{th}\) probe = \(h(k) \mod \text{TableSize}\)
1\(^{th}\) probe = \((h(k) + f(1)) \mod \text{TableSize}\)
2\(^{th}\) probe = \((h(k) + f(2)) \mod \text{TableSize}\)

\[
\ldots
\]

\(i^{th}\) probe = \((h(k) + f(i)) \mod \text{TableSize}\)

Apply probing for both insert and find...
Need consistency to find where you previously inserted
Terminology Alert!

Separate chaining is sometimes called open hashing.

Open addressing is sometimes called closed hashing.
Open Addressing Example, Revisited

Insert:
38
19
8
109
10

Try \( h(K) \)
If full, try \( h(K) + 1 \).
If full, try \( h(K) + 2 \).
If full, try \( h(K) + 3 \).
Etc…

What is \( f(i) \)?
Linear Probing

\[ f(i) = i \]

- Probe sequence:
  \[ 0^{th} \text{ probe} = h(K) \mod \text{TableSize} \]
  \[ 1^{th} \text{ probe} = (h(K) + 1) \mod \text{TableSize} \]
  \[ 2^{th} \text{ probe} = (h(K) + 2) \mod \text{TableSize} \]
  \[ \ldots \]
  \[ i^{th} \text{ probe} = (h(K) + i) \mod \text{TableSize} \]
Linear Probing – Clustering

no collision

no collision

collision in small cluster

collision in large cluster
Analysis of Linear Probing

• For any $\lambda < 1$, linear probing will find an empty slot
• Expected # of probes (for large table sizes)
  – unsuccessful search:
    
    \[
    \frac{1}{2} \left( 1 + \frac{1}{(1 - \lambda)^2} \right)
    \]
  – successful search:
    
    \[
    \frac{1}{2} \left( 1 + \frac{1}{1 - \lambda} \right)
    \]

• Linear probing suffers from primary clustering
• Performance quickly degrades for $\lambda > 1/2$
Quadratic Probing

\[ f(i) = i^2 \]

- Probe sequence:
  
  \[
  \begin{align*}
  0^{\text{th}} \text{ probe} &= h(K) \% \text{ TableSize} \\
  1^{\text{th}} \text{ probe} &= (h(K) + 1) \% \text{ TableSize} \\
  2^{\text{th}} \text{ probe} &= (h(K) + 4) \% \text{ TableSize} \\
  3^{\text{th}} \text{ probe} &= (h(K) + 9) \% \text{ TableSize} \\
  \cdots \\
  i^{\text{th}} \text{ probe} &= (h(K) + i^2) \% \text{ TableSize}
  \end{align*}
  \]
Quadratic Probing Example

Insert:
89
18
49
58
79
Quadratic Probing Example

Insert: 89 18 49 58
Quadratic Probing Example

Insert:
89
18
49
58
79
Quadratic Probing Example

Insert:
89
18
49
58
79
Quadratic Probing Example

Insert:
89
18
49
58
79
Quadratic Probing Example

Insert: 89 18 49 58 79

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>49</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>79</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>18</td>
<td></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>
<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>
<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>
<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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>89</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>89</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Another Quadratic Probing Example

Table Size = 7
\( h(K) = K \mod 7 \)

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

- insert(76) \( 76 \mod 7 = 6 \)
- insert(40) \( 40 \mod 7 = 5 \)
- insert(48) \( 48 \mod 7 = 6 \)
- insert(5) \( 5 \mod 7 = 5 \)
- insert(55) \( 55 \mod 7 = 6 \)
- insert(47) \( 47 \mod 7 = 5 \)
Another Quadratic Probing Example

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>2</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>
</tr>
<tr>
<td>4</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>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>76</td>
</tr>
</tbody>
</table>

TableSize = 7
h(K) = K % 7

- insert(76) 76 % 7 = 6
- insert(40) 40 % 7 = 5
- insert(48) 48 % 7 = 6
- insert(5)  5 % 7 = 5
- insert(55) 55 % 7 = 6
- insert(47) 47 % 7 = 5
Another Quadratic Probing Example

Table Size = 7
h(K) = K % 7

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>76</td>
</tr>
</tbody>
</table>

- insert(76) 76 % 7 = 6
- insert(40) 40 % 7 = 5
- insert(48) 48 % 7 = 6
- insert(5) 5 % 7 = 5
- insert(55) 55 % 7 = 6
- insert(47) 47 % 7 = 5
Another Quadratic Probing Example

TableSize = 7
h(K) = K \% 7

<table>
<thead>
<tr>
<th></th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>76</td>
</tr>
</tbody>
</table>

- insert(76)  76 \% 7 = 6
- insert(40)  40 \% 7 = 5
- insert(48)  48 \% 7 = 6
- insert(5)   5 \% 7 = 5
- insert(55)  55 \% 7 = 6
- insert(47)  47 \% 7 = 5
Another Quadratic Probing Example

TableSize = 7
\( h(K) = K \mod 7 \)

- insert(76) \( 76 \mod 7 = 6 \)
- insert(40) \( 40 \mod 7 = 5 \)
- insert(48) \( 48 \mod 7 = 6 \)
- insert(5) \( 5 \mod 7 = 5 \)
- insert(55) \( 55 \mod 7 = 6 \)
- insert(47) \( 47 \mod 7 = 5 \)
Another Quadratic Probing Example

TableSize = 7
\( h(K) = K \mod 7 \)

- insert(76) \( 76 \mod 7 = 6 \)
- insert(40) \( 40 \mod 7 = 5 \)
- insert(48) \( 48 \mod 7 = 6 \)
- insert(5) \( 5 \mod 7 = 5 \)
- insert(55) \( 55 \mod 7 = 6 \)
- insert(47) \( 47 \mod 7 = 5 \)
**Another Quadratic Probing Example**

<table>
<thead>
<tr>
<th>Table Size = 7</th>
<th>h(K) = K % 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert(76)</td>
<td>76 % 7 = 6</td>
</tr>
<tr>
<td>insert(40)</td>
<td>40 % 7 = 5</td>
</tr>
<tr>
<td>insert(48)</td>
<td>48 % 7 = 6</td>
</tr>
<tr>
<td>insert(5)</td>
<td>5 % 7 = 5</td>
</tr>
<tr>
<td>insert(55)</td>
<td>55 % 7 = 6</td>
</tr>
<tr>
<td>insert(47)</td>
<td>47 % 7 = 5</td>
</tr>
</tbody>
</table>

Can’t insert 47, even though the table is not full.

---

0 + 5 = 5 % 7 = 5  
1 + 5 = 6 % 7 = 6  
4 + 5 = 9 % 7 = 2  
9 + 5 = 14 % 7 = 0  
16 + 5 = 21 % 7 = 0  
25 + 5 = 30 % 7 = 2  
36 + 5 = 41 % 7 = 6  
49 + 5 = 54 % 7 = 5  
64 + 5 = 69 % 7 = 6  

......
Quadratic Probing: 
Success guarantee for $\lambda < \frac{1}{2}$

Assertion #1:
If T = TableSize is prime and $\lambda < \frac{1}{2}$, then quadratic probing will find an empty slot in $T/2$ probes or fewer.

Assertion #2:
For prime T and all $0 \leq i, j \leq T/2$ and $i \neq j$,

$$(h(K) + i^2) \% T \neq (h(K) + j^2) \% T$$

(prove this by contradiction if you’d like)

Assertion #3: Assertion #2 proves assertion #1.
Quadratic Probing: Properties

• For any $\lambda < \frac{1}{2}$, quadratic probing will find an empty slot; for bigger $\lambda$, quadratic probing may find a slot.

• Quadratic probing does not suffer from primary clustering: keys hashing to the same area are not bad.

• But what about keys that hash to the same spot?
  – Secondary Clustering!
Double Hashing

Idea: given two different (good) hash functions $h(K)$ and $g(K)$, it is unlikely two keys to collide with both.

So...let’s try probing with a second hash function:

$$f(i) = i \times g(K)$$

• Probe sequence:
  
  0\textsuperscript{th} probe = $h(K) \mod \text{TableSize}$
  
  1\textsuperscript{th} probe = $(h(K) + g(K)) \mod \text{TableSize}$
  
  2\textsuperscript{th} probe = $(h(K) + 2 \times g(K)) \mod \text{TableSize}$
  
  3\textsuperscript{th} probe = $(h(K) + 3 \times g(K)) \mod \text{TableSize}$
  
  \ldots
  
  i\textsuperscript{th} probe = $(h(K) + i \times g(K)) \mod \text{TableSize}$
Double Hashing Example

TableSize = 7
h(K) = K % 7
g(K) = 5 – (K % 5)

- Insert(76) 76 % 7 = 6 and 5 - (76 % 5) = 4
- Insert(93) 93 % 7 = 2 and 5 - (93 % 5) = 2
- Insert(40) 40 % 7 = 5 and 5 - (40 % 5) = 5
- Insert(47) 47 % 7 = 5 and 5 - (47 % 5) = 3
- Insert(10) 10 % 7 = 3 and 5 - (10 % 5) = 5
- Insert(55) 55 % 7 = 6 and 5 - (55 % 5) = 5
Double Hashing Example

TableSize = 7
h(K) = K % 7
g(K) = 5 – (K % 5)

Insert(76) 76 % 7 = 6 and 5 - (76 % 5) = 4
Insert(93) 93 % 7 = 2 and 5 - (93 % 5) = 2
Insert(40) 40 % 7 = 5 and 5 - (40 % 5) = 5
Insert(47) 47 % 7 = 5 and 5 - (47 % 5) = 3
Insert(10) 10 % 7 = 3 and 5 - (10 % 5) = 5
Insert(55) 55 % 7 = 6 and 5 - (55 % 5) = 5
Double Hashing Example

Table Size = 7

\( h(K) = K \mod 7 \)

\( g(K) = 5 - (K \mod 5) \)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>2</td>
<td></td>
<td>93</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>
</tr>
<tr>
<td>4</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>
</tr>
<tr>
<td>6</td>
<td>76</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Insert(76) \( 76 \mod 7 = 6 \) and \( 5 - (76 \mod 5) = 4 \)

Insert(93) \( 93 \mod 7 = 2 \) and \( 5 - (93 \mod 5) = 2 \)

Insert(40) \( 40 \mod 7 = 5 \) and \( 5 - (40 \mod 5) = 5 \)

Insert(47) \( 47 \mod 7 = 5 \) and \( 5 - (47 \mod 5) = 3 \)

Insert(10) \( 10 \mod 7 = 3 \) and \( 5 - (10 \mod 5) = 5 \)

Insert(55) \( 55 \mod 7 = 6 \) and \( 5 - (55 \mod 5) = 5 \)
Double Hashing Example

TableSize = 7
h(K) = K \% 7
g(K) = 5 - (K \% 5)

Insert(76) 76 \% 7 = 6 and 5 - (76 \% 5) = 4
Insert(93) 93 \% 7 = 2 and 5 - (93 \% 5) = 2
Insert(40) 40 \% 7 = 5 and 5 - (40 \% 5) = 5
Insert(47) 47 \% 7 = 5 and 5 - (47 \% 5) = 3
Insert(10) 10 \% 7 = 3 and 5 - (10 \% 5) = 5
Insert(55) 55 \% 7 = 6 and 5 - (55 \% 5) = 5
Double Hashing Example

Table Size = 7

\[
\begin{align*}
\text{h}(K) &= K \mod 7 \\
\text{g}(K) &= 5 - (K \mod 5) 
\end{align*}
\]

<table>
<thead>
<tr>
<th>Insert</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>47</td>
<td>93</td>
<td>47</td>
</tr>
<tr>
<td>2</td>
<td>93</td>
<td>93</td>
<td>93</td>
</tr>
<tr>
<td>3</td>
<td>76</td>
<td>76</td>
<td>76</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>47</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>55</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>55</td>
<td>55</td>
<td>55</td>
</tr>
</tbody>
</table>

Insert(76) \( 76 \mod 7 = 6 \) and \( 5 - (76 \mod 5) = 4 \)
Insert(93) \( 93 \mod 7 = 2 \) and \( 5 - (93 \mod 5) = 2 \)
Insert(40) \( 40 \mod 7 = 5 \) and \( 5 - (40 \mod 5) = 5 \)
Insert(47) \( 47 \mod 7 = 5 \) and \( 5 - (47 \mod 5) = 3 \)
Insert(10) \( 10 \mod 7 = 3 \) and \( 5 - (10 \mod 5) = 5 \)
Insert(55) \( 55 \mod 7 = 6 \) and \( 5 - (55 \mod 5) = 5 \)
Double Hashing Example

TableSize = 7
h(K) = K % 7
g(K) = 5 – (K % 5)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>93</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>76</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Insert(76)  76 % 7 = 6  and  5 - (76 % 5) = 4
Insert(93)  93 % 7 = 2  and  5 - (93 % 5) = 2
Insert(40)  40 % 7 = 5  and  5 - (40 % 5) = 5
Insert(47)  47 % 7 = 5  and  5 - (47 % 5) = 3
Insert(10)  10 % 7 = 3  and  5 - (10 % 5) = 5
Insert(55)  55 % 7 = 6  and  5 - (55 % 5) = 5
Double Hashing Example

Table Size = 7
h(K) = K % 7
g(K) = 5 - (K % 5)

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>47</td>
</tr>
<tr>
<td>2</td>
<td>93</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>55</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>76</td>
</tr>
</tbody>
</table>

- Insert(76) 76 % 7 = 6 and 5 - (76 % 5) = 4
- Insert(93) 93 % 7 = 2 and 5 - (93 % 5) = 2
- Insert(40) 40 % 7 = 5 and 5 - (40 % 5) = 5
- Insert(47) 47 % 7 = 5 and 5 - (47 % 5) = 3
- Insert(10) 10 % 7 = 3 and 5 - (10 % 5) = 5
- Insert(55) 55 % 7 = 6 and 5 - (55 % 5) = 5
Analysis of Double Hashing

• Double hashing is safe for $\lambda < 1$ for at least one case:
  – $h(k) = k \mod p$
  – $g(k) = q - (k \mod q)$
  – $2 < q < p$, and $p$, $q$ are primes

• Expected # of probes (for large table sizes)
  – unsuccessful search:
    $$\frac{1}{1 - \lambda}$$
  – successful search:
    $$\frac{1}{\lambda} \log_e \left( \frac{1}{1 - \lambda} \right)$$
Deletion in Separate Chaining

How do we delete an element with separate chaining?

Easy, just delete the item from the bucket
Deletion in Open Addressing

Can we do something similar for open addressing?

- Delete
- Find
- Insert

h(k) = k \% 7
Linear probing

Delete(23)
Find(59)
Insert(30)
Deletion in Open Addressing

Can we do something similar for open addressing?

- Delete
- Find
- Insert

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

Linear probing

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>16</td>
<td></td>
<td>59</td>
<td></td>
<td>76</td>
</tr>
</tbody>
</table>

Delete(23)
Find(59)
Insert(30)
Deletion in Open Addressing

Can we do something similar for open addressing?

- Delete
- Find
- Insert

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

Linear probing

- Delete(23)
- Find(59)
- Insert(30)

Need to leave a marker of a deletion
Deletion in Open Addressing

Can we do something similar for open addressing?

- Delete
- Find
- Insert

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

Linear probing

- Delete(23)
- Find(59)
- Insert(30)

Need to leave a marker of a deletion
**Rehashing**

**Idea:** When the table gets too full, create a bigger table (usually 2x as large) and hash all the items from the original table into the new table.

- **When to rehash?**
  - Separate chaining: full ($\lambda = 1$)
  - Open addressing: half full ($\lambda = 0.5$)
  - When an insertion fails
  - Some other threshold

- **Cost of a single rehashing?** $O(N)$
Rehashing Example

- Separate chaining example:
  \[ h_1(x) = x \mod 5 \] rehashes to \[ h_2(x) = x \mod 11. \]

\[
\begin{array}{cccc}
\lambda = 1 & 0 & 1 & 2 \\
& 25 & 37 & 83 \\
& 52 & 98 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
\lambda = 5/11 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
& 25 & 37 & 83 & 52 & 98 \\
\end{array}
\]
Rehashing Picture

- Starting with table of size 2, double when load factor > 1.

Amortized analysis can show cost of inserting m keys < 3m
Discussion on Hashing

• Hash tables are one of the most important data structures.

• Hashing has many applications where operations are limited to find, insert, and delete.

• Can use either separate chaining or open hashing
  — Java uses separate chaining

• Rehashing has good amortized complexity.

• Also has a big data version to minimize disk accesses: extendible hashing. (See textbook.)
Java hashCode() Method

• Class Object defines a hashCode( ) method
  – Intent: returns a suitable hashcode for the object
  – Result is arbitrary int; must scale to fit a hash table
    (e.g. obj.hashCode() % nBuckets)
  – Used by collection classes like HashMap

• Classes should override with calculation appropriate for instances of the class
  – Calculation should involve semantically “significant” fields of objects
hashCode() and equals()

- To work correctly, particularly with collection classes (like HashMap), an Object’s hashCode() and equals() must obey this rule:
  
  if a.equals(b) then it must be true that a.hashCode() == b.hashCode()

- Why? Is the reverse also required?
Aside: Properties of Mod

To keep hashed values within the size of the table, we will generally do:

\[ h(K) = \text{function}(K) \% \text{TableSize} \]

It’s worth noting a couple properties of the mod function:

- \((a + b) \% c = [(a \% c) + (b \% c)] \% c\)
- \((a \times b) \% c = [(a \% c) \times (b \% c)] \% c\)
- \(a \% c = b \% c \rightarrow (a - b) \% c = 0\)
Designing Hash Functions

We’ve seen a few possibilities. The simplest is **modular hashing**:

\[ h(K) = K \% P \]

where \( P \) is usually just the TableSize.

\( P \) is often chosen to be prime:

- Reduces likelihood of collisions due to patterns in data
- Is useful for guarantees on certain hashing strategies (as we’ll see)

But what would be a more convenient value of \( P \)?
Amortized Analysis of Rehashing

• Cost of inserting n keys is < 3n
• \(2^k + 1 \leq n \leq 2^{k+1}\)
  – Hashes = n
  – Rehashes = \(2 + 2^2 + \ldots + 2^k = 2^{k+1} - 2\)
  – Total = \(n + 2^{k+1} - 2 < 3n\)
• Example
  – \(n = 33\), Total = \(33 + 64 - 2 = 95 < 99\)