**Today’s Outline**

- **Announcements**
  - Project 3 partner selection due Mon Feb 22 by 11pm, DO NOT WAIT UNTIL THEN TO START!
  - Written Homework #5 due Friday 2/19

- **Today’s Topics:**
  - Hash Tables
  - Disjoint Sets

---

### Quadratic Probing Example

<table>
<thead>
<tr>
<th>Insertion</th>
<th>Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>76%7 = 6</td>
<td>0</td>
</tr>
<tr>
<td>40%7 = 5</td>
<td>1</td>
</tr>
<tr>
<td>48%7 = 6</td>
<td>2</td>
</tr>
<tr>
<td>5%7 = 5</td>
<td>3</td>
</tr>
<tr>
<td>55%7 = 6</td>
<td>4</td>
</tr>
</tbody>
</table>

- **insert(76)**
- **insert(40)**
- **insert(48)**
- **insert(5)**
- **insert(55)**

- **insert(47)**
  - 47%7 = 5

---

**Quadratic Probing: Properties**

- For any \( \lambda < \frac{1}{2} \), quadratic probing will find an empty slot; for bigger \( \lambda \), quadratic probing may not 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**!

---

**Quadratic Probing: Success guarantee for \( \lambda < \frac{1}{2} \)**

- If size is prime and \( \lambda < \frac{1}{2} \), then quadratic probing will find an empty slot in size/2 probes or fewer.
  - For all \( 0 \leq i, j \leq \text{size}/2 \) and \( i \neq j \)
    - \( (h(x) + i^2) \mod \text{size} \neq (h(x) + j^2) \mod \text{size} \)
  - By contradiction: suppose that for some \( i \neq j \):
    - \( (h(x) + i^2) \mod \text{size} = (h(x) + j^2) \mod \text{size} \)
    - \( i^2 \mod \text{size} = j^2 \mod \text{size} \)
    - \( (i^2 - j^2) \mod \text{size} = 0 \)
    - \( [(i + j)(i - j)] \mod \text{size} = 0 \)
    - BUT size does not divide \((i-j)\) or \((i+j)\)

---

**Double Hashing**

\[ f(i) = i \times g(k) \]

where \( g \) is a second hash function.

- Probe sequence:
  - 0\(^{th} \) probe = \( h(k) \mod \text{TableSize} \)
  - 1\(^{st} \) probe = \( (h(k) + g(k)) \mod \text{TableSize} \)
  - 2\(^{nd} \) probe = \( (h(k) + 2g(k)) \mod \text{TableSize} \)
  - 3\(^{rd} \) probe = \( (h(k) + 3g(k)) \mod \text{TableSize} \)
  - ... \( i^{th} \) probe = \( (h(k) + ig(k)) \mod \text{TableSize} \)
Double Hashing Example

\[ i^{th} \text{ probe} = (h(k) + i \times g(k)) \mod \text{TableSize} \]
\[ h(k) = k \mod 7 \text{ and } g(k) = 5 - (k \mod 5) \]

<table>
<thead>
<tr>
<th>i</th>
<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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>47</td>
<td>47</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Hashing Summary

- Hashing is one of the most important data structures.
- Hashing has many applications where operations are limited to find, insert, and delete.
- Dynamic hash tables have good amortized complexity.

Resolving Collisions with Double Hashing

<table>
<thead>
<tr>
<th>Hash Functions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ H(K) = K \mod M ]</td>
</tr>
<tr>
<td>[ H_2(K) = 1 + \left( \lfloor K/M \rfloor \mod (M-1) \right) ]</td>
</tr>
<tr>
<td>[ M = ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Insert these values into the hash table in this order. Resolve any collisions with double hashing:</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 28 33 147 43</td>
</tr>
</tbody>
</table>

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?
  - half full (\( \lambda = 0.5 \))
  - when an insertion fails
  - some other threshold
- Cost of rehashing?