Set implementation: insert

• Similar structure to contains
  – Calculate hash of new element
  – Check if the element is already in the set

• Add the element to the front of the list that is at table[hash(value)]
```java
public boolean add(String value) {
    int valuePosition = hash(value);

    // check to see if the value is already in the set
    StringHashEntry temp = table[valuePosition];
    while (temp != null) {
        if (temp.data.equals(value)) {
            return false;
        }
        temp = temp.next;
    }

    // add the value to the set
    StringHashEntry newEntry = new StringHashEntry(value, table[valuePosition]);
    table[valuePosition] = newEntry;
    size++;
    return true;
}
```
public boolean remove(String value) {
    int valuePosition = hash(value);
    if (table[valuePosition] == null) { // empty bucket
        return false;
    }
    if (table[valuePosition].data.equals(value)) { // removing front
        table[valuePosition] = table[valuePosition].next;
        size--; return true;
    }
    StringHashEntry temp = table[valuePosition];
    while (temp.next != null) { // find value
        if (temp.next.data.equals(value)) {
            temp.next = temp.next.next;
            size--; return true;
        }
        temp = temp.next;
    }
    return false;
}
Hash versus tree

- Which is better, a hash set or a tree set?

<table>
<thead>
<tr>
<th>Hash</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Implementing Set ADT (Revisited)

<table>
<thead>
<tr>
<th></th>
<th>Insert</th>
<th>Remove</th>
<th>Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted array</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Sorted array</td>
<td>$\Theta(\log(n) + n)$</td>
<td>$\Theta(\log(n) + n)$</td>
<td>$\Theta(\log(n))$</td>
</tr>
<tr>
<td>Linked list</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>BST (if balanced)</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>Hash table</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>
Probing hash tables

• Alternative strategy for collision resolution: try alternative cells until empty cell found
  – cells $h_0(x), h_1(x), h_2(x), \ldots$ tried in succession,
    where $h_i(x) = (\text{hash}(x) + f(i)) \% \text{TableSize}$
  – $f$ is collision resolution strategy
  – bigger table needed
Linear probing

• **linear probing**: resolving collisions in slot \( i \) by putting the colliding element into the next available slot \((i+1, i+2, \ldots)\)
  
  – add 41, 34, 7, 18, then 21, then 57
    • 21 collides (41 is already there), so we search ahead until we find empty slot 2
    • 57 collides (7 is already there), so we search ahead twice until we find empty slot 9

  – lookup algorithm becomes slightly modified; we have to loop now until we find the element or an empty slot
    • what happens when the table gets mostly full?
Linear probing

• $f(i) = i$

• Probe sequence:
  
  $0^{th}$ probe = $h(x) \mod TableSize$
  
  $1^{th}$ probe = $(h(x) + 1) \mod TableSize$
  
  $2^{th}$ probe = $(h(x) + 2) \mod TableSize$
  
  $\ldots$
  
  $i^{th}$ probe = $(h(x) + i) \mod TableSize$
Primary clustering problem

- **clustering**: nodes being placed close together by probing, which degrades hash table's performance
  - add 89, 18, 49, 58, 9
  - now searching for the value 28 will have to check half the hash table! no longer constant time...
Linear probing – clustering

- No collision
- Collision in small cluster
- Collision in large cluster
Alternative probing strategy

• Primary clustering occurs with linear probing because the same linear pattern:
  – if a slot is inside a cluster, then the next slot must either:
    • also be in that cluster, or
    • expand the cluster

• Instead of searching forward in a linear fashion, consider searching forward using a quadratic function
Quadratic probing

- **quadratic probing**: resolving collisions on slot $i$ by putting the colliding element into slot $i+1$, $i+4$, $i+9$, $i+16$, ...
  - add 89, 18, 49, 58, 9
    - 49 collides (89 is already there), so we search ahead by +1 to empty slot 0
    - 58 collides (18 is already there), so we search ahead by +1 to occupied slot 9, then +4 to empty slot 2
    - 9 collides (89 is already there), so we search ahead by +1 to occupied slot 0, then +4 to empty slot 3
  - what is the lookup algorithm?
Quadratic probing in action

\[
\begin{align*}
\text{hash ( 89, 10 )} &= 9 \\
\text{hash ( 18, 10 )} &= 8 \\
\text{hash ( 49, 10 )} &= 9 \\
\text{hash ( 58, 10 )} &= 8 \\
\text{hash ( 9, 10 )} &= 9
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>After insert 89</th>
<th>After insert 18</th>
<th>After insert 49</th>
<th>After insert 58</th>
<th>After insert 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>
</tr>
<tr>
<td>2</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>
</tr>
<tr>
<td>4</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>
</tr>
<tr>
<td>6</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>
</tr>
<tr>
<td>8</td>
<td></td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
</tbody>
</table>
Quadratic probing

• $f(i) = i^2$

• Probe sequence:
  
  0\textsuperscript{th} probe = $h(x) \mod TableSize$
  
  1\textsuperscript{st} probe = (h(x) + 1) \mod TableSize
  
  2\textsuperscript{nd} probe = (h(x) + 4) \mod TableSize
  
  3\textsuperscript{rd} probe = (h(x) + 9) \mod TableSize
  
  \ldots

  $i\textsuperscript{th}$ probe = $(h(x) + i^2) \mod TableSize
Quadratic probing benefit

• If one of $h + i^2$ falls into a cluster, this does not imply the next one will

• For example, suppose an element was to be inserted in bucket 23 in a hash table with 31 buckets
  – The sequence in which the buckets would be checked is:
    23, 24, 27, 1, 8, 17, 28, 10, 25, 11, 30, 20, 12, 6, 2, 0
Quadratic probing benefit

• Even if two buckets are initially close, the sequence in which subsequent buckets are checked varies greatly
  – Again, with TableSize = 31, compare the first 16 buckets which are checked starting with elements 22 and 23:

  22  22, 23, 26, 0, 7, 16, 27, 9, 24, 10, 29, 19, 11, 5, 1, 30
  23  23, 24, 27, 1, 8, 17, 28, 10, 25, 11, 30, 20, 12, 6, 2, 0

• Quadratic probing solves the problem of primary clustering
Quadratic probing drawbacks

- Suppose we have 8 buckets:
  \[ 1^2 \mod 8 = 1, \ 2^2 \mod 8 = 4, \ 3^2 \mod 8 = 1 \]
  - In this case, we are checking bucket \( h(x) + 1 \) twice having checked only one other bucket

- No guarantee that
  \[ (h(x) + i^2) \mod TableSize \]
  will cycle through 0, 1, ..., \( TableSize - 1 \)
Quadratic probing

• Solution:
  – require that $TableSize$ be prime
  – $(h(x) + i^2) \mod TableSize$ for $i = 0, ..., (TableSize - 1)/2$ will cycle through $(TableSize + 1)/2$ values before repeating

• Example with $M = 11$:
  $0, 1, 4, 9, 16 \equiv 5, 25 \equiv 3, 36 \equiv 3$

• With $M = 13$:
  $0, 1, 4, 9, 16 \equiv 3, 25 \equiv 12, 36 \equiv 10, 49 \equiv 10$

• With $M = 17$:
  $0, 1, 4, 9, 16, 25 \equiv 8, 36 \equiv 2, 49 \equiv 15, 64 \equiv 13, 81 \equiv 13$

Note: the symbol $\equiv$ means "$\mod M =$"
Double hashing

- **double hashing**: resolve collisions on slot \( i \) by applying a second hash function

- \( f(i) = i \times g(x) \)
  where \( g \) is a second hash function
  - limitations on what \( g \) can evaluate to?
  - recommended: \( g(x) = R - (x \mod R) \), where \( R \) prime smaller than \( TableSize \)

- Probe sequence:
  - 0\(^{th} \) probe = \( h(x) \mod TableSize \)
  - 1\(^{th} \) probe = \( (h(x) + g(x)) \mod TableSize \)
  - 2\(^{th} \) probe = \( (h(x) + 2 \times g(x)) \mod TableSize \)
  - 3\(^{th} \) probe = \( (h(x) + 3 \times g(x)) \mod TableSize \)
  - \( \ldots \)
  - \( i^{th} \) probe = \( (h(x) + i \times g(x)) \mod TableSize \)
Double Hashing Example

\[ h(x) = x \mod 7 \text{ and } g(x) = 5 - (x \mod 5) \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>Probes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>41</td>
<td>16</td>
<td>40</td>
<td>47</td>
<td>10</td>
<td>55</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>16</td>
<td></td>
<td>47</td>
<td>16</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>16</td>
<td></td>
<td>16</td>
<td>10</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>2</td>
</tr>
</tbody>
</table>

Probes 1