Overflow Handling

- An overflow occurs when the home bucket for a new pair \((\text{key}, \text{element})\) is full.
- We may handle overflows by:
  - Search the hash table in some systematic fashion for a bucket that is not full.
    - Linear probing (linear open addressing).
    - Quadratic probing.
    - Random probing.
  - Eliminate overflows by permitting each bucket to keep a list of all pairs for which it is the home bucket.
    - Array linear list.
    - Chain.

Linear Probing – Get And Put

- \(\text{divisor} = b \) (number of buckets) = 17.
- \(\text{Home bucket} = \text{key} \% 17\).

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
</tr>
</tbody>
</table>

- Put in pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45
Linear Probing – Remove

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
</tr>
</tbody>
</table>

- remove(0)

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
</tr>
</tbody>
</table>

- Search cluster for pair (if any) to fill vacated bucket.

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
</tr>
</tbody>
</table>

Linear Probing – remove(34)

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
</tr>
</tbody>
</table>

- Search cluster for pair (if any) to fill vacated bucket.

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
<td>7</td>
</tr>
</tbody>
</table>
**Linear Probing – remove(29)**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
<td>7</td>
<td>28</td>
</tr>
<tr>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
<td>7</td>
<td>28</td>
</tr>
<tr>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
<td>7</td>
<td>28</td>
</tr>
<tr>
<td>0</td>
<td>45</td>
<td>6</td>
<td>23</td>
<td>7</td>
<td>28</td>
</tr>
</tbody>
</table>

- Search cluster for pair (if any) to fill vacated bucket.

**Performance Of Linear Probing**

- Worst-case get/put/remove time is $\Theta(n)$, where $n$ is the number of pairs in the table.
- This happens when all pairs are in the same cluster.
Expected Performance

- \( \text{alpha} = \text{loading density} = \frac{\text{number of pairs}}{b} \).
  - \( \text{alpha} = \frac{12}{17} \).
- \( S_n \) = expected number of buckets examined in a successful search when \( n \) is large
- \( U_n \) = expected number of buckets examined in an unsuccessful search when \( n \) is large
- Time to put and remove governed by \( U_n \).

\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c}
\hline
\text{0} & \text{4} & \text{8} & \text{12} & \text{16} \\
\hline
\text{34} & \text{0} & \text{45} & \text{6} & \text{23} & \text{7} & \text{28} & \text{12} & \text{29} & \text{11} & \text{30} & \text{33} \\
\hline
\end{array}

\begin{itemize}
  
  \item \( S_n \approx \frac{1}{2}(1 + \frac{1}{1-\text{alpha}}) \)
  
  \item \( U_n \approx \frac{1}{2}(1 + \frac{1}{(1-\text{alpha})^2}) \)
  
  \item Note that \( 0 \leq \text{alpha} \leq 1 \).
\end{itemize}

\begin{tabular}{|c|c|c|}
\hline
\text{alpha} & \text{S_n} & \text{U_n} \\
\hline
\text{0.50} & 1.5 & 2.5 \\
\hline
\text{0.75} & 2.5 & 8.5 \\
\hline
\text{0.90} & 5.5 & 50.5 \\
\hline
\end{tabular}

Alpha \( \leq 0.75 \) is recommended.
Hash Table Design

• Performance requirements are given, determine maximum permissible loading density.

• We want a successful search to make no more than 10 compares (expected).
  - $S_n \sim \frac{1}{2}(1 + \frac{1}{1 - \alpha})$
  - $\alpha \leq \frac{18}{19}$

• We want an unsuccessful search to make no more than 13 compares (expected).
  - $U_n \sim \frac{1}{2}(1 + \frac{1}{(1 - \alpha)^2})$
  - $\alpha \leq \frac{4}{5}$

• So $\alpha \leq \min\{\frac{18}{19}, \frac{4}{5}\} = \frac{4}{5}$.

Hash Table Design

• Dynamic resizing of table.
  - Whenever loading density exceeds threshold ($\frac{4}{5}$ in our example), rehash into a table of approximately twice the current size.

• Fixed table size.
  - Know maximum number of pairs.
  - No more than 1000 pairs.
  - Loading density $\leq \frac{4}{5} \Rightarrow b \geq \frac{5}{4} \times 1000 = 1250$.
  - Pick $b$ (equal to divisor) to be a prime number or an odd number with no prime divisors smaller than 20.
Linear List Of Synonyms

- Each bucket keeps a linear list of all pairs for which it is the home bucket.
- The linear list may or may not be sorted by key.
- The linear list may be an array linear list or a chain.

Sorted Chains

- Put in pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45
- Home bucket = key % 17
Expected Performance

- Note that $\alpha \geq 0$.
- Expected chain length is $\alpha$.
- $S_n \sim 1 + \alpha/2$.
- $U_n \leq \alpha$, when $\alpha < 1$.
- $U_n \sim 1 + \alpha/2$, when $\alpha \geq 1$.

java.util.Hashtable

- Unsorted chains.
- Default initial $b = \text{divisor} = 101$
- Default $\alpha \leq 0.75$
- When loading density exceeds max permissible density, rehash with $\text{newB} = 2b+1$. 