Overflow Handling

- An overflow occurs when the home bucket for a new pair (key, 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

- divisor = b (number of buckets) = 17.
- Home bucket = key % 17.

<table>
<thead>
<tr>
<th>0  4  8  12 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ 0  45 ] [ 6  23 7 ] [ 28 12 29 11 30 33 ]</td>
</tr>
</tbody>
</table>

- Put in pairs whose keys are 6, 12, 29, 28, 11, 23, 7, 0, 33, 30, 45

Linear Probing – Remove

- remove(0)

<table>
<thead>
<tr>
<th>0  4  8  12 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ 0  45 ] [ 6  23 7 ] [ 28 12 29 11 30 33 ]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>0  4  8  12 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ 0  45 ] [ 6  23 7 ] [ 28 12 29 11 30 33 ]</td>
</tr>
</tbody>
</table>

Linear Probing – remove(34)

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

<table>
<thead>
<tr>
<th>0  4  8  12 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ 0  45 ] [ 6  23 7 ] [ 28 12 29 11 30 33 ]</td>
</tr>
</tbody>
</table>

Linear Probing – remove(29)

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

<table>
<thead>
<tr>
<th>0  4  8  12 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ 0  45 ] [ 6  23 7 ] [ 28 12 29 11 30 33 ]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>alpha</th>
<th>S_n</th>
<th>U_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.50</td>
<td>1.5</td>
<td>2.5</td>
</tr>
<tr>
<td>0.75</td>
<td>2.5</td>
<td>8.5</td>
</tr>
<tr>
<td>0.90</td>
<td>5.5</td>
<td>50.5</td>
</tr>
</tbody>
</table>

Alpha <= 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 \( \alpha \leq \min \{\frac{18}{19}, \frac{4}{5}\} = \frac{4}{5}. \)

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 \( \text{newB} = 2b+1 \).