CSCI 241
Scott Wehrwein
Hash Tables: Rehashing
Goals

Know how and why to grow and shrink the capacity of a hash table by resizing the array and rehashing its contents.

Be prepared to implement rehashing so it runs in worst-case $O(C + n)$.
Load Factor: Performance Implications

Load factor $\lambda = \frac{\# \text{ entries in table}}{\text{size of the array}}$

If $\lambda$ is large, runtime is slow. If $\lambda$ is small, memory is wasted.

**Strategy:** grow or shrink array when $\lambda$ gets too large or small.
Shrinking the array

Need to **rehash**: put each element where it belongs in the new array.

<table>
<thead>
<tr>
<th>0</th>
<th>10 “bear”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 “dog” → 11 “auk”</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>14 “cat” → 24 “ape”</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></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

(10 % 3) -> 1
(1 % 3) -> 1
(11 % 3) -> 2
(14 % 3) -> 2
(24 % 3) -> 0
Rehashing: Runtime, take 1

Rehashing algorithm:
for each bucket $b$:
  for each element $e$ in $b$:
    put $e$ into the new array

Let $C =$ array size
Let $n =$ number of entries

Overall runtime is:
• worst-case $O(C + n^2)$
• average-case $O(C + n)$

visits $C$ buckets
visits $n$ entries (total)
could be $O(n) =$
Rehashing: Runtime, take 2

Rehashing algorithm:
for each bucket \( b \):
  for each element \( e \) in \( b \):
    put \( e \) into the new array

Let \( C = \) array size
Let \( n = \) number of entries

Overall runtime is:
• worst-case \( O(C + n) \)

visits \( C \) buckets
visits \( n \) entries (total)

could it be \( O(n) \)?

put is \( O(n) \) because it has to search for existing keys.
Here, we can’t have duplicate keys: all entries were already in the map!

Consequence: we don’t need to search the bucket when rehashing