CS 225
Data Structures

March 26 – Hash Table Collisions
Wade Fagen-Ulmschneider
Collision Handling: Separate Chaining

\( S = \{ 16, 8, 4, 13, 29, 11, 22 \} \quad |S| = n \)

\( h(k) = k \% 7 \quad |Array| = N \)

**Worst Case**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst Case</th>
<th>SUHA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>Remove/Find</td>
<td>( O(n) )</td>
<td>( O(\alpha) )</td>
</tr>
</tbody>
</table>
Collision Handling: Probe-based Hashing

$S = \{ 16, 8, 4, 13, 29, 11, 22 \}$  \hspace{1cm} |S| = n

$h(k) = k \% 7$  \hspace{1cm} |Array| = N

(Example of closed hashing)
Collision Handling: Linear Probing

\[ S = \{ 16, 8, 4, 13, 29, 11, 22 \} \]

| Array | = N

\[ h(k) = k \mod 7 \]

Try \( h(k) = (k + 0) \mod 7 \), if full...

Try \( h(k) = (k + 1) \mod 7 \), if full...

Try \( h(k) = (k + 2) \mod 7 \), if full...

Try ...

<table>
<thead>
<tr>
<th>Worst Case</th>
<th>SUHA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td></td>
</tr>
<tr>
<td>Remove/Find</td>
<td></td>
</tr>
</tbody>
</table>
A Problem w/ Linear Probing

Primary clustering:

Description:

Remedy:
Collision Handling: Double hashing

$S = \{16, 8, 4, 13, 29, 11, 22\}$  $|S| = n$

$h(k) = k \mod 7$  $|Array| = N$

Try $h(k) = (k + 0 \times h_2(k)) \mod 7$, if full...
Try $h(k) = (k + 1 \times h_2(k)) \mod 7$, if full...
Try $h(k) = (k + 2 \times h_2(k)) \mod 7$, if full...
Try ...

$h(k, i) = (h_1(k) + i \times h_2(k)) \mod 7$
Running Times

The expected number of probes for find(key) under SUHA

**Linear Probing:**
- Successful: \( \frac{1}{2}(1 + 1/(1-\alpha)) \)
- Unsuccessful: \( \frac{1}{2}(1 + 1/(1-\alpha))^2 \)

**Double Hashing:**
- Successful: \( \frac{1}{\alpha} \times \ln(1/(1-\alpha)) \)
- Unsuccessful: \( 1/(1-\alpha) \)

**Separate Chaining:**
- Successful: \( 1 + \alpha/2 \)
- Unsuccessful: \( 1 + \alpha \)

*(Don’t memorize these equations, no need.)*

Instead, observe:
- As \( \alpha \) increases:
- If \( \alpha \) is constant:
Running Times

The expected number of probes for find(key) under SUHA

**Linear Probing:**
- Successful: $\frac{1}{2}(1 + \frac{1}{(1-\alpha)})$
- Unsuccessful: $\frac{1}{2}(1 + \frac{1}{(1-\alpha)})^2$

**Double Hashing:**
- Successful: $\frac{1}{\alpha} \times \ln(\frac{1}{(1-\alpha)})$
- Unsuccessful: $\frac{1}{(1-\alpha)}$
ReHashing

What if the array fills?
Which collision resolution strategy is better?

• Big Records:

• Structure Speed:

What structure do hash tables replace?

What constraint exists on hashing that doesn’t exist with BSTs?

Why talk about BSTs at all?
## Running Times

<table>
<thead>
<tr>
<th></th>
<th>Hash Table</th>
<th>AVL</th>
<th>Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Find</strong></td>
<td>Amortized:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Worst Case:</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Insert</strong></td>
<td>Amortized:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Worst Case:</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Storage Space</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
std data structures

std::map
std data structures

\texttt{std::map}:
\begin{itemize}
  \item \texttt{operator[]} \quad & \text{Insert an element.} \\
  \item \texttt{insert} \quad & \text{Insert an element.} \\
  \item \texttt{erase} \quad & \text{Remove an element.} \\
\end{itemize}

\begin{itemize}
  \item \texttt{lower\_bound(key)} \quad \rightarrow \quad \text{Iterator to first element } \leq \text{ key} \\
  \item \texttt{upper\_bound(key)} \quad \rightarrow \quad \text{Iterator to first element } > \text{ key}
\end{itemize}
std data structures

std::unordered_map
  ::operator[]
  ::insert
  ::erase

    ::lower_bound(key) → Iterator to first element ≤ key
    ::upper_bound(key) → Iterator to first element > key
std data structures

std::unordered_map
   ::operator[]
   ::insert
   ::erase

   ::lower_bound(key) \rightarrow Iterator to first element \leq key
   ::upper_bound(key) \rightarrow Iterator to first element > key

   ::load_factor()
   ::max_load_factor(ml) \rightarrow Sets the max load factor
Secret, Mystery Data Structure

ADT:
insert
remove
isEmpty