CS 225

Data Structures

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

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

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

(Worst Case)

(Example of open hashing)
Collision Handling: Probe-based Hashing

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

\[ h(k) = k \mod 7 \quad |\text{Array}| = N \]
Collision Handling: Linear Probing

\[ S = \{ 16, 8, 4, 13, 29, 11, 22 \} \quad |S| = n \]
\[ h(k) = k \mod 7 \quad |Array| = N \]

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 ...

(Example of closed hashing)
A Problem w/ Linear Probing

Primary clustering:

Description:

Remedy:
Collision Handling: Double hashing

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

\[ h(k) = k \% 7 \quad |\text{Array}| = N \]

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

\[ h(k, i) = (h_1(k) + i \times h_2(k)) \% 7 \]
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: \( \frac{1}{\alpha} \ln(\frac{1}{1-\alpha}) \)
- Unsuccessful: \( \frac{1}{1-\alpha} \)

**Separate Chaining:**
- Successful: \( 1 + \frac{\alpha}{2} \)
- 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: \( \frac{1}{2}(1 + 1/(1-\alpha)) \)
• Unsuccessful: \( \frac{1}{2}(1 + 1/(1-\alpha))^2 \)

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

std::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 \text{Iterator to first element } \leq \text{key}
  ::upper_bound(key) \rightarrow \text{Iterator to first element } > \text{key}
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`
- `::load_factor()`
- `::max_load_factor(ml) → Sets the max load factor`
Secret, Mystery Data Structure

ADT:
insert
remove
isEmpty