Team Contract and Proposal Due April 9th

Team Contract:

Be sure to ‘sign’ electronically.

Non-participants may be removed from groups!

Project Proposal:

One of your three algorithms should be completed by mid-project check-in.
Learning Objectives

• Review fundamentals of hash tables

• Introduce closed hashing approaches to hash collisions

• Determine when and how to resize a hash table

• Justify when to use different index approaches

• If time: General strategies for creating a hash function
A Hash Table based Dictionary

Client Code:

```
1 | Dictionary<KeyType, ValueType> d;
2 | d[k] = v;
```

A **Hash Table** consists of three things:
1. A hash function
2. A data storage structure
3. A method of addressing *hash collisions*
### Insertion (Separate Chaining)

#### Table:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>B+</td>
<td>2</td>
</tr>
<tr>
<td>Anna</td>
<td>A-</td>
<td>4</td>
</tr>
<tr>
<td>Alice</td>
<td>A+</td>
<td>4</td>
</tr>
<tr>
<td>Betty</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>Brett</td>
<td>A-</td>
<td>2</td>
</tr>
<tr>
<td>Greg</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>Sue</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>Ali</td>
<td>B+</td>
<td>4</td>
</tr>
<tr>
<td>Laura</td>
<td>A</td>
<td>7</td>
</tr>
<tr>
<td>Lily</td>
<td>B+</td>
<td>7</td>
</tr>
</tbody>
</table>

#### Diagram:

- **Key**: Greg, Brett, Betty, Bob, Anna, Alice, Sue, Ali, Laura, Lily
- **Value**: A, B, A+, A-, B+, B+ (separate chaining)
- **Hash**: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Hash Table (Separate Chaining)

For hash table of size $m$ and $n$ elements:

find runs in: __________.

insert runs in: __________.

remove runs in: __________.
Simple Uniform Hashing Assumption

Given table of size $m$, a simple uniform hash, $h$, implies

$$\forall k_1, k_2 \in U \text{ where } k_1 \neq k_2, \ Pr(h[k_1] = h[k_2]) = \frac{1}{m}$$

**Uniform:** keys are equally likely to hash to any position

**Independent:** key hash values are independent of other keys
Separate Chaining Under SUHA

Under SUHA, a hash table of size $m$ and $n$ elements:

Expected length of chain is ________________.
Separate Chaining Under SUHA

Under SUHA, a hash table of size $m$ and $n$ elements:

- find runs in: __________.
- insert runs in: __________.
- remove runs in: __________.
Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

- **Open Hashing**: store \(k,v\) pairs externally

- **Closed Hashing**: store \(k,v\) pairs in the hash table
Collision Handling: Probe-based Hashing

$S = \{ 1, 8, 15 \}$

$h(k) = k \mod 7$

$|S| = n$

$|\text{Array}| = m$
Collision Handling: Linear Probing

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

\[ h(k, i) = (k + i) \% 7 \]

Try \( h(k) = (k + 0) \% 7 \), if full...
Try \( h(k) = (k + 1) \% 7 \), if full...
Try \( h(k) = (k + 2) \% 7 \), if full...
Try ...
Collision Handling: Linear Probing

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

- \text{find}(29)
- \text{remove}(16)

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

Primary clustering:

Description:

Remedy:
Collision Handling: Quadratic Probing

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

\[ |S| = n \]

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

\[ |Array| = m \]

\[ h(k, i) = (k + i \times i) \mod 7 \]

Try \( h(k) = (k + 0) \mod 7 \), if full...
Try \( h(k) = (k + 1 \times 1) \mod 7 \), if full...
Try \( h(k) = (k + 2 \times 2) \mod 7 \), if full...
Try ...
A Problem w/ Quadratic Probing

Secondary clustering:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0'</th>
<th>0''</th>
</tr>
</thead>
</table>

Description:

Remedy:
Collision Handling: Double Hashing

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

\[ |S| = n \]

\[ |\text{Array}| = m \]

\[ h_1(k) = k \mod 7 \]

\[ h_2(k) = 5 - (k \mod 5) \]

\[ h(k, i) = (h_1(k) + i \times h_2(k)) \mod 7 \]

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 ...
Running Times

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

*(Expectation under SUHA)*

**Open Hashing:**

insert: ____________.

find/ remove: ____________.  

**Closed Hashing:**

insert: ____________.

find/ remove: ____________.  

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

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: $\frac{1}{\alpha} \times \ln\left(\frac{1}{1-\alpha}\right)$
- Unsuccessful: $\frac{1}{1-\alpha}$

**Separate Chaining:**
- Successful: $1 + \frac{\alpha}{2}$
- Unsuccessful: $1 + \alpha$

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: $\frac{1}{\alpha} \ln(\frac{1}{1-\alpha})$
- 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 implement?

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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amortized:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Worst Case:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Insert</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amortized:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Worst Case:</td>
<td></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 \leq key
  ::upper_bound(key) → Iterator to first element > 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
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
Bonus Slides
Hash Function (Division Method)

Hash of form: \( h(k) = k \% m \)

Pro:

Con:
Hash Function (Multiplication Method)

Hash of form: $h(k) = \lfloor m(kA \mod 1) \rfloor$, $0 \leq A \leq 1$

Pro:

Con:
Hash Function (Universal Hash Family)

Hash of form: \( h_{ab}(k) = ((ak + b) \mod p) \mod m, \ a, b \in \mathbb{Z}_p^*, \mathbb{Z}_p \)

\( \forall k_1 \neq k_2, \ Pr_{a,b}(h_{ab}[k_1] = h_{ab}[k_2]) \leq \frac{1}{m} \)

Pro:

Con: