Hashing (cont.) + Wrap-Up

Hitesh Boinpally
Summer 2023
Agenda

- General Reminders
- Hashing (cont.)
- Victory Lap
Agenda

• General Reminders
• Hashing (cont.)
• Victory Lap
General Reminders

• Final exam next Wednesday (8/16) and Friday (8/18)
  • Seating chart posted
  • Practice Exams posted (solutions will be posted tonight)
• Review session next Monday (8/14), 10:30 – 1:00pm
  • TA-led in GUG 220
  • Will give another practice exam (will be recorded and posted too)
• Resub 5 due tonight
• Resub 6 (last one) due next Friday
• Course evals will open soon
  • Important to hear what went well and what can improve!
Agenda

• General Reminders
• Hashing (cont.)
• Victory Lap
Hashing

- **Idea**: Map every value for some object to some integer index
  - Store these values in an array based on the index (**hash table**)
- **Hash Function**: An algorithm to do this mapping
Hashing

• **Idea**: Map every value for some object to some integer index
  • Store these values in an array based on the index (**hash table**)

• **Hash Function**: An algorithm to do this mapping
  • Requirements:
    • The same object should always have the same number
    • If two objects are considered “equal” they should have the same hash code
  • To be **good**:
    • Results should be distributed approximately uniformly
    • Should “look random”
Collisions

• **Collision**: When hash function maps 2 values to same index
• **Collision Resolution**: An algorithm for fixing collisions

```java
set.add(11);
set.add(49);
set.add(24);
set.add(7);
set.add(54);
set.add(54); // collides with 24! Where should it go?
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>49</td>
</tr>
</tbody>
</table>
Probing

- **Probing**: Resolving a collision by moving to another index
  - **Linear Probing**: Moves to the next index

```java
set.add(11);
set.add(49);
set.add(24);
set.add(7);
set.add(54);  // collides with 24; utilize probing
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>49</td>
</tr>
</tbody>
</table>
Probing

- **Probing**: Resolving a collision by moving to another index
  - **Linear Probing**: Moves to the next index

```java
set.add(11);
set.add(49);
set.add(24);
set.add(7);
set.add(54);  // collides with 24; utilize probing
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>54</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>49</td>
</tr>
</tbody>
</table>

Lesson 15 - Summer 2023
Probing

• **Probing**: Resolving a collision by moving to another index
  • **Linear Probing**: Moves to the next index

```java
set.add(11);
set.add(49);
set.add(24);
set.add(7);
set.add(54); // collides with 24; utilize probing
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>54</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>49</td>
</tr>
</tbody>
</table>

• Another version of this is **Quadratic Probing** – moves to indices by squares (increasingly far away)
Clustering

- **Clustering**: Clumps of elements at neighboring indexes
  - slows down the hash table lookup; you must loop through them

```java
set.add(11);
set.add(49);
set.add(24);
set.add(7);
set.add(54); // collides with 24; utilize probing
set.add(14); // collides with 24, then 54
set.add(86); // collides with 14, then 7
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>54</td>
<td>14</td>
<td>7</td>
<td>86</td>
<td>49</td>
</tr>
</tbody>
</table>
Clustering

- **Clustering**: Clumps of elements at neighboring indexes
  - slows down the hash table lookup; you must loop through them
    
    ```
    set.add(11);
    set.add(49);
    set.add(24);
    set.add(7);
    set.add(54);  // collides with 24; utilize probing
    set.add(14);  // collides with 24, then 54
    set.add(86);  // collides with 14, then 7
    ```

    | index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
    |-------|---|---|---|---|---|---|---|---|---|---|
    | value | 0 | 11| 0 | 0 | 24| 54| 14| 7 | 86| 49|

- Think about what would happen if we were checking 94 now?
Alternative: Chaining

- **Chaining**: Resolving collisions by storing a list at each index
  - add/search/remove must traverse lists, but the lists are short
  - impossible to "run out" of indexes

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>54</td>
<td>14</td>
<td>7</td>
<td>86</td>
<td>49</td>
</tr>
</tbody>
</table>
Alternative: Chaining

- **Chaining**: Resolving collisions by storing a list at each index
  - add/search/remove must traverse lists, but the lists are short
  - impossible to "run out" of indexes

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>24</td>
<td>86</td>
<td>7</td>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
14
```

```
54
```

```
11  24  86  7  49
     ^           ^
     |           |
     54          14
```
Rehashing

- **Rehashing**: Growing to a larger array when the table is too full
  - Cannot simply copy the old array to a new one
- **Load Factor**: ratio of (# of elements) / (hash table length)
  - Many collections rehash when load factor ≅ ¾
  - Big prime numbers as hash table sizes help reduce collisions
Hash Maps

• A hash map is a set, but with key, value pairs as elements:

```java
// key     value
map.put("Marty", 14);
map.put("Jeff", 21);
map.put("Kasey", 20);
map.put("Stef", 35);
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
"Stef" 35  "Marty" 14  "Jeff" 21  "Kasey" 20
```

• Instead of a List<Integer>, write an inner Entry node class with key and value fields; the map stores a List<Entry>
Agenda

• General Reminders
• Hashing (cont.)
• Victory Lap
Learning Objectives

or, “What did I learn in this class?”

Seven themes:

• Computational Thinking
• Code Comprehension
• Code Writing
• Communication

• Testing
• Debugging
• Ethics/Impact
Applications of CS

or “What can I do with what I learned?”

• Detect and prevent toxicity online
• Digitize basketball players
• Help DHH people identify sounds
• Figure out how to best distribute relief funds
• Recognize disinformation online
• Make movies
• Improve digital collaboration
• Fix Olympic badminton
• And so much more!
Future Courses
or “What can I do next?”

CSE Majors

<table>
<thead>
<tr>
<th>Course</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSE 311</td>
<td>Mathematical foundations</td>
</tr>
<tr>
<td>CSE 351</td>
<td>Low-level computer organization/abstraction</td>
</tr>
<tr>
<td>CSE 331</td>
<td>Software design/implementation</td>
</tr>
<tr>
<td>CSE 340</td>
<td>Interaction programming</td>
</tr>
<tr>
<td>CSE 341</td>
<td>Programming languages (!)</td>
</tr>
</tbody>
</table>

Non-CSE Majors/Open to All (*)

<table>
<thead>
<tr>
<th>Course</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSE 154*</td>
<td>Intro. to web programming (several languages)</td>
</tr>
<tr>
<td>CSE 163*</td>
<td>Intermediate programming, data analysis (Python)</td>
</tr>
<tr>
<td>CSE 180*</td>
<td>Introduction to data science (Python)</td>
</tr>
<tr>
<td>CSE 373</td>
<td>Data structures and algorithms</td>
</tr>
<tr>
<td>CSE 374</td>
<td>Low-level programming and tools (C/C++)</td>
</tr>
<tr>
<td>CSE 412</td>
<td>Data Visualization</td>
</tr>
<tr>
<td>CSE 416</td>
<td>Intro. to Machine Learning</td>
</tr>
</tbody>
</table>

Frequently Asked Questions

• How can I get better at programming?
  • Practice!

• How can I learn to X?
  • Search online, read books, look at examples

• What should I work on next?
  • Anything you can think of! (Here are some ideas)
  • Beware: it’s hard to tell what’s easy and what’s hard.

• Should I learn another language? Which one?
  • That depends—what do you want to do?

• What’s the best programming language?
Thank you!!!