CSE 123 Summer 2024

LEC 15: Hashing

Questions during Class?
Raise hand or send here
sli.do    #cse123

BEFORE WE START

Music: 123 24su Lecture Tunes 🌞

Instructor: Joe Spaniac
TAs: Andras Daniel, Eric Nicole, Sahej Trien, Zach

YOUR CODE IS WITHOUT A DOUBT THE WORST I HAVE EVER RUN
BUT IT DOES RUN
Lecture Outline

• Announcements

• Hashing

• Programming HashSet

• Final Remarks
Announcements

• P4 Spam Classifier Released!!!
  - Topics: Recursion, Machine Learning (AI)
  - PLEASE read the Specification (we can’t emphasize this enough)

• Resubmission Period 6 closes tonight, 8/09 at 11:59pm

Upcoming…

• Sections Next Tuesday/Thursday will be Exam Review!
• Final Exam: Friday (8/16) 10:50-11:50am (GWN 301)
Lecture Outline

• Announcements

• Hashing
  - Introduction
  - Rules of Hash Codes
  - What is it and how do we use it?
  - Example

• Collisions and Resolving Them

• Programming HashSet

• Final Remarks
HashMaps and Hash Sets

- We can store our values here in a Hash Table
- These efficiently store and retrieve data
- How do we make Hash Table operations fast?
- Through the use of *Hashing*

<table>
<thead>
<tr>
<th>Operation</th>
<th>ArrayList</th>
<th>LinkedList</th>
<th>HashTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains(x)</td>
<td>O(n)</td>
<td>O(n)</td>
<td></td>
</tr>
<tr>
<td>add(x)</td>
<td>O(1)*</td>
<td>O(1)</td>
<td></td>
</tr>
<tr>
<td>remove(x)</td>
<td>O(n)</td>
<td>O(n)</td>
<td></td>
</tr>
</tbody>
</table>

* take higher level CS courses if you’re curious about the ambiguity
# HashMaps and Hash Sets

- We can store our values here in a Hash Table
- These efficiently store and retrieve data
- How do we make Hash Table operations fast?
- Through the use of *Hashing*

<table>
<thead>
<tr>
<th>Operation</th>
<th>ArrayList</th>
<th>LinkedList</th>
<th>HashTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains(x)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(1) (average), O(n) (worst)</td>
</tr>
<tr>
<td>add(x)</td>
<td>O(1)*</td>
<td>O(1)</td>
<td>O(1) (average), O(n) (worst)</td>
</tr>
<tr>
<td>remove(x)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(1) (average), O(n) (worst)</td>
</tr>
</tbody>
</table>

*take higher level CS courses if you’re curious about the ambiguity*
Basic Idea of Hashing

- Converting data (our values or objects) into a **Hash Code**
- **Hash Code** is basically like a “fingerprint”
- We store our data under the **Hash Code** *(fingerprint)*
- Use this **Hash Code** to retrieve our data
- Motivation: Makes searching, storing, and getting values easier, especially in large databases

![Diagram showing Input and "Fingerprint"]
What is a Hash Function?

Hash Function - An algorithm that produces an “fingerprint” of the original message

“hello world” → Hashing Function → Output: 42

- Other names: Hash or Hash Code, Fingerprint, Checksum, Digest, …
What makes a good Hash Function?

Equal “Objects” should always result in the same Hash Code.

- “hello world” → Hashing Function → Output: 42
- “hello mom” → Hashing Function → Output: 98

* Can’t be random
What is an Example Hash Function?

For Integers

\[ h(x) = x \mod \text{size} \]
Recap

Let’s Store our value “What is the meaning of life”

User: “Hash Function, what is my Hash Code?”

Computer: “Your Hash Code is: ” __________

User: “Ok, let’s store our value under this fingerprint”

Computer: “Sure can do Chief”
Recap

Let’s Store our value “What is the meaning of life”

User: “Can I get my value back?”

Computer: “No”

User: “???”

User: “Why not?!?”

Computer: “What is the hash code?”

User: “42”

Computer: “Okie, here you go!”

Returns: “What is the meaning of life”

User: “Why are you so difficult 😭”
What is an Example Hash Function?

Let’s say for Integer \(x\)

\[ h(x) = x \mod \text{size} \]
What about this Example?

Let’s say for Integer $x$

$h(x) = x \% \text{size}$

<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>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>56</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lecture Outline

• Announcements

• Hashing

• Collisions and Resolving Them
  - Linear Probing
  - Quadratic Probing
  - ⭐ Chaining

• Programming HashSet

• Final Remarks
What is Linear Probing?

A way to resolve collisions by adding the element in the next available spot

Regular Hash Function

\[ h(x) = x \% \text{size} \]

Hash Function (if collision)

\[ h'(x) = [h(x) + f(i)] \% \text{size} \]

\[ f(i) = i \]
What is Linear Probing?

\[ h(x) = x \% \text{size} \quad h'(x) = [h(x) + f(i)] \% \text{size} \quad f(i) = i \]
What is Quadratic Probing?

A way to resolve collisions by adding the element in the next available spot (quadratically)

Regular Hash Function
\[ h(x) = x \% \text{size} \]

Hash Function (if collision)
\[ h'(x) = [h(x) + f(i)] \% \text{size} \]
\[ f(i) = i^2 \]
What is Quadratic Probing?

\[ h(x) = x \mod \text{size} \quad h'(x) = [h(x) + f(i)] \mod \text{size} \quad f(i) = i^2 \]
What is ⭐Chaining?

A way to resolve collisions by creating a LinkedList at that Index (also called a “bucket”)

• Combines both features of ArrayList Indexing and the ease of adding values using LinkedLists
What is Chaining?

\[ h(x) = x \% \text{size} \]
# Recap (Comparison)

<table>
<thead>
<tr>
<th>Linear Probing</th>
<th>Quadratic Probing</th>
<th>Chaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>A way to resolve collisions by adding the element in the next available spot</td>
<td>A way to resolve collisions by adding the element in the next available spot (quadratically)</td>
<td>A way to resolve collisions by creating a LinkedList at that Index (also called a “bucket”)</td>
</tr>
</tbody>
</table>
Why ⭐Chaining?

Clustering - A tendency for data to clump together when using solutions to Collisions like Linear and Quadratic probing

- Linear and Quadratic Probing often result in “Clustering”
- Inefficient use of space in the table
- This means the Runtimes will also be slower
## Why ⭐ Chaining?

<table>
<thead>
<tr>
<th>Operation</th>
<th>ArrayList</th>
<th>LinkedList</th>
<th>HashTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains(x)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(1) (average), O(n) (worst)</td>
</tr>
<tr>
<td>add(x)</td>
<td>O(1)*</td>
<td>O(1)</td>
<td>O(1) (average), O(n) (worst)</td>
</tr>
<tr>
<td>remove(x)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(1) (average), O(n) (worst)</td>
</tr>
</tbody>
</table>

- Hashing can reduce it down to **O(1)**
- “Load Factor” - lambda (\( \lambda \))
  - the number of values in each LinkedList
- Finding the index in the Table is **O(1)**
- Finding value in LinkedList is **O(\( \lambda \))** or essentially **O(1)**
Lecture Outline

• Announcements

• Hashing

• Collisions and Resolving Them

• **Programming HashSet**

• Final Remarks
Lecture Outline

• Announcements
• Hashing
• Collisions and Resolving Them
• Programming HashSet

• Final Remarks