# 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

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

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

# Probing

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

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

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

# Probing

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

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

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

# Probing

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

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

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

  - 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

```
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** |

# 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?

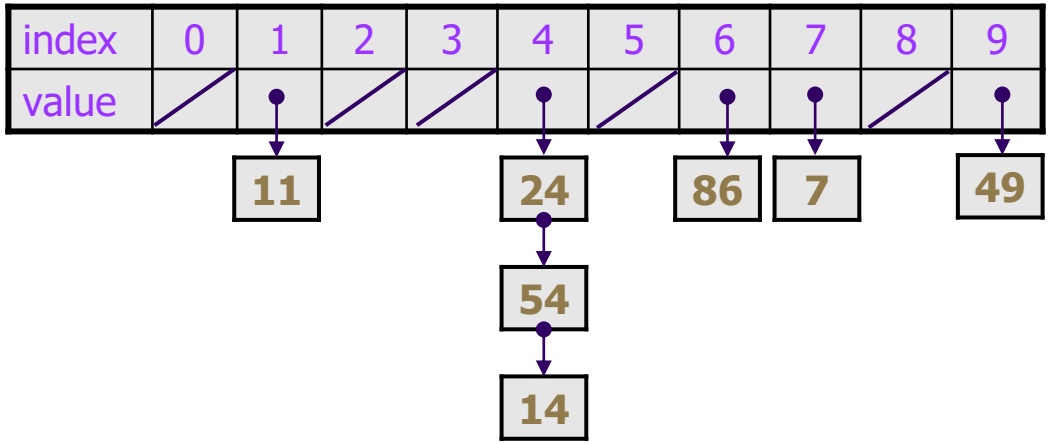
# Alternative: Chaining

- **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

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

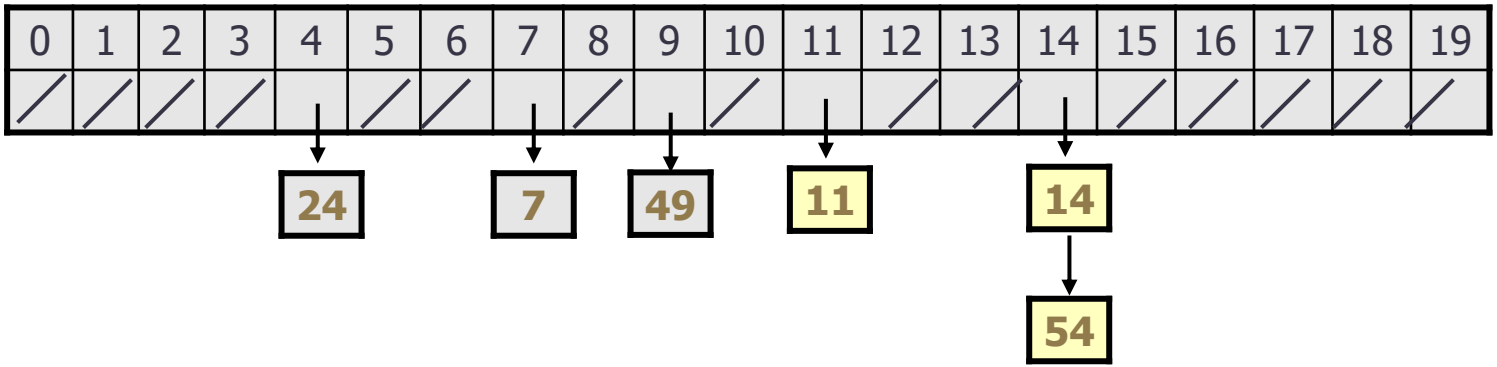
# Alternative: Chaining

- **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



# Rehashing

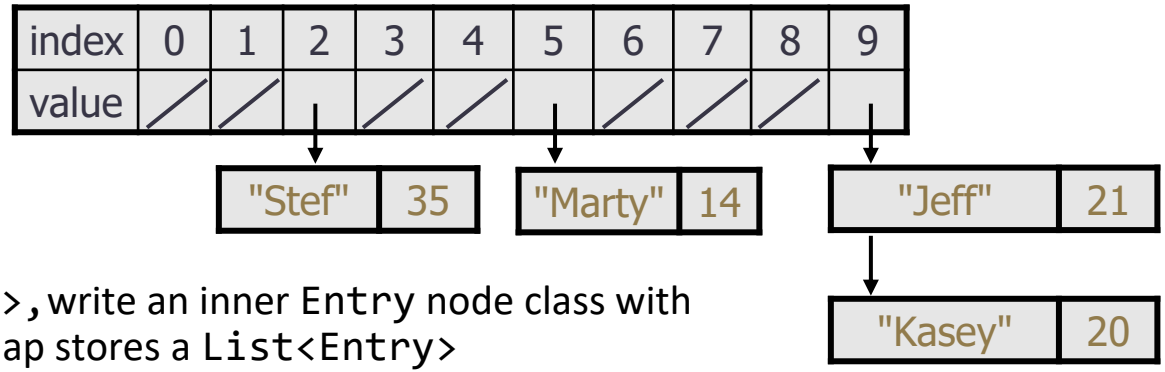
- **Rehashing**: Growing to a larger array when the table is too full
  - Cannot simply copy the old array to a new one
- **Load Factor:** ratio of (*# of elements* ) / (*hash table length* )
  - Many collections rehash w when load factor ≅ ¾
  - Big prime numbers as hash table sizes help reduce collisions



# Hash Maps

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

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




- 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**

| Course | Overview |
|---|---|
| CSE 311 | Mathematical foundations |
| CSE 351 | Low-level computer organization/abstraction |
| CSE 331 | Software design/implementation |
| CSE 340 | Interaction programming |
| CSE 341 | Programming languages (!) |

**Non-CSE Majors/Open to All (*)**

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

*See:* https://www.cs.washington.edu/academics/ugrad/current-students and https://www.cs.washington.edu/academics/ugrad/nonmajor-options/nonmajor-courses

# 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!!!