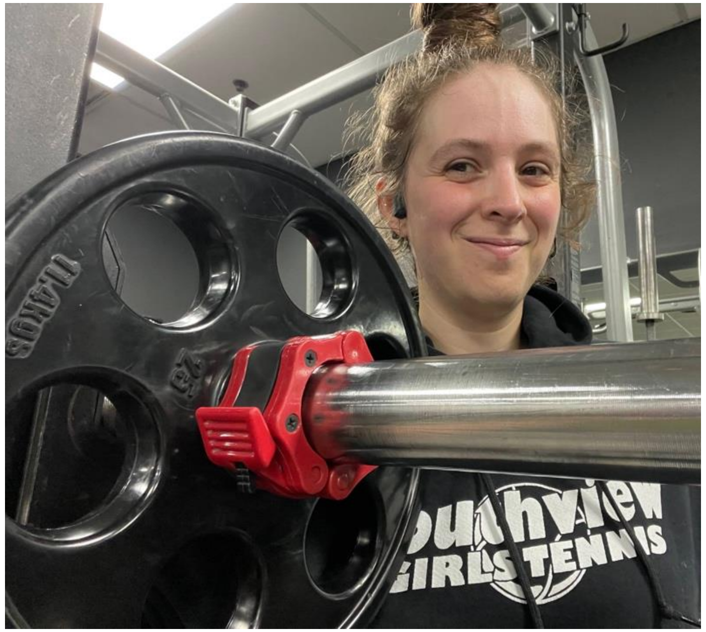
Lecture 19: Sets and Maps

CS 2110, Matt Eichhorn and Leah Perlmutter

October 30, 2025

Announcements

- Process Over Product
  - I don’t lift weights because I want the weights to be in the air!
  - Keep track of the real goals of learning – the changes in yourself.
- Remember self care to support your learning.



Announcements

- Teaching and Learning Philosophy: Questions and silences
  - when I ask a question, it's not to get the answer as fast as possible, but rather to give everyone a chance to think about the question

Announcements

- Coming soon: Ed post about Prelim 2

Roadmap

Java, Complexity, OOP

- start– 9/30

ADTs I

- List, Stack, Queue, Iteration
  - 10/2 – 10/16

ADTs II

- Trees
  - Tues 10/21, Thurs 10/23, Thurs 10/28
- **Set and Map**
  - **Today**
- Hash Tables
  - Tues 11/4
- Graphs
  - Tues 11/6, Thurs 11/11, Tues 11/13

Beyond ADTs

- Graphical User Interfaces, Parallel Programming
  - 11/18 – end of semester

Overview of today

Sets

- Set Basics
- Set Performance by Representation
- Set Implementation (Live coding)

Maps

- Map Basics
- Map Implementation
- Dynamic Priority Queues

Set Basics

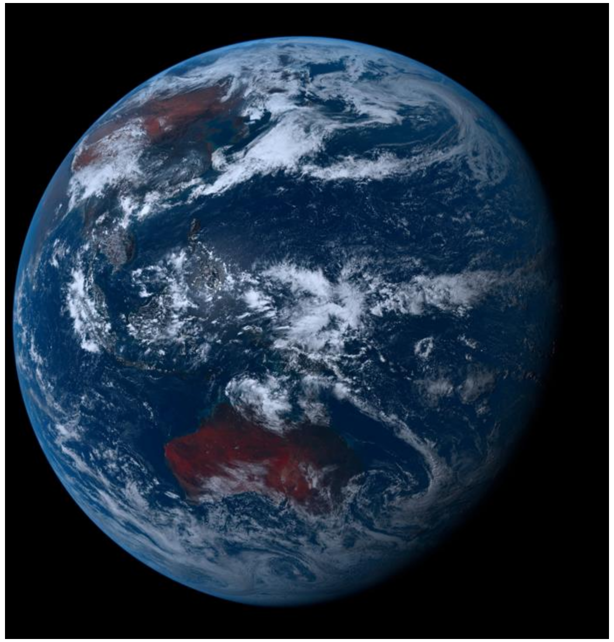
Set Basics

- Unordered
- Distinct elements

Examples

- Finite Set (mathematics)
- Set of countries in the world
- …

{2, 1, 3}



Set Operations

- **Contains** – is the element in the set?
  - parameter: element
  - return type: boolean
- **Add** – add element to set if not already there. Did we succeed?
  - parameter: element
  - return type: boolean
- **Remove** – remove element from set if exists. Did we succeed?
  - parameter: element
  - return type: boolean
- **Size –** how many elements are in this set?
  - return type: int

Overview of today

Sets

- Set Basics ✅
- Set Performance by Representation
- Set Implementation (Live coding)

Maps

- Map Basics
- Map Implementation
- Dynamic Priority Queues

Set Performance

Set Performance

Representation	Contains	Add	Remove	Size
Unordered List	O(N)	O(N)	O(N)	O(1)
Ordered List	O(log N)	O(N)	O(N)	O(1)
Balanced Tree	O(log N)	O(log N)	O(log N)	O(1)
???	O(1)*	O(1)*	O(1)*	O(1)

Overview of today

Sets

- Set Basics ✅
- Set Performance by Representation ✅
- Set Implementation (Live coding)

Maps

- Map Basics
- Map Implementation
- Dynamic Priority Queues

Set Implementation

Pollev (for use during demo)

PollEv.com/leahp

text `leahp` to 22333

Code Demo

Overview of today

Sets

- Set Basics ✅
- Set Performance by Representation ✅
- Set Implementation (Live coding) ✅

Maps

- Map Basics
- Map Implementation
- Dynamic Priority Queues

Map Basics

The Map ADT

- Given a word, return its definition
- Given a contact's name, return their phone number
- Given a student's ID number, return their NetID, course history, student records, etc.

- All involve looking up a value associated with some key

- Also known as:
  - Dictionary
  - Associative array

Map Operations

- put
  - parameters: key, value
  - Associate *value* with *key*
- get
  - parameter: key
  - Return value associated with *key*
- remove
  - parameter: key
  - Remove any association with *key,* if it existed, and return it
- containsKey
  - parameter: key
  - Return true if an association exists with that key, false if not
- size
  - Return number of associations in the Map

Map interface

Generic on two type parameters

```
interface Map<K, V> {…}
```

- K: Type of keys
- V: Type of values

Key/Value pair is an Entry

```
record Entry<K, V> (K key, V value) {...}
```

- Similar to a set of entries
- Keys are unique (a key can only map to one value)

Interface methods

`void put(K key,  V value)`
Associate *value* with *key*

`V get(K key)`
Return value associated with *key*

`V remove(K key)`
Remove any association for *key*

`Set<K> keySet()`
Allow iterating over keys

`boolean containsKey(K key)`
Tell if key is in map

Exercise

- Let `bdays` be a `Map<String, LocalDate>` that associates people's names with their birthday
- Write a loop to print the name and birthday of everyone in `bdays`

`void put(K key, V value)`
Associate *value* with *key*

`V get(K key)`
Return value associated with *key*

`V remove(K key)`
Remove any association for *key*

`Set<K> keySet()`
Allow iterating over keys

`boolean containsKey(K key)`
Tell if key is in map

Example client code

```
Map<String, LocalDate> bdays = …;

bdays.put("Alan Turing", LocalDate.of(1912, 6, 23));
bdays.put("John von Neumann", LocalDate.of(1903, 12, 28));

println("Alan Turing was born on " + bdays.get("Alan Turing"));
```

```
for (String name : bdays.keySet()) {
    println(name + " was born on " + bdays.get(name));
}
```

```
println("Do I know Barbara Liskov's birthday? " +
        (bdays.containsKey("Barbara Liskov") ? "yes" : "no"));
```

Map Performance

Representation	Contains	Put	Get	Size
Unordered List	O(N)	O(N)	O(N)	O(1)
Ordered List	O(log N)	O(N)	O(N)	O(1)
Balanced Tree	O(log N)	O(log N)	O(log N)	O(1)
???	O(1)*	O(1)*	O(1)*	O(1)

Overview of today

Sets

- Set Basics ✅
- Set Performance by Representation ✅
- Set Implementation (Live coding) ✅

Maps

- Map Basics ✅
- Map Implementation
- Dynamic Priority Queues

Map Implementation

Set versus Map code

```
// Set
@Override
public boolean contains(T elem) {
    assert elem != null;
    for (T member : this.data) {
        if (member.equals(elem)) {
            return true;
        }
    }
    return false;
}
```

```
// Map (if we wrote contains from scratch)
@Override
public boolean containsKey(K key) {
    assert key != null;
    for (Entry<K, V> entry : this.data) {
        if (entry.key().equals(key)) {
            return true;
        }
    }
    return false;
}
```

Take Away: While conceptually Map is similar to a Set of Entry, Map cannot simply be a Set<Entry<K, V>> because uniqueness in a Set is based on no repeated *elements* whereas in a Map, it's based on no repeated *keys*. The Map code has to delve into the Entry and check/manipulate its key, whereas Set code manipulates whole elements.

Overview of today

Sets

- Set Basics ✅
- Set Performance by Representation ✅
- Set Implementation (Live coding) ✅

Maps

- Map Basics ✅
- Map Implementation ✅
- Dynamic Priority Queues

Dynamic Priority Queue

Dynamic Priority Queue

```java
/**
 * A dynamic priority queue implementation using composition with a max heap.
 */
public class MaxHeapDynamicPriorityQueue<T> implements DynamicPriorityQueue<T> {

    /**
     * Represents an association of a `priority` to an `elem`.
     *`Entry`s are compared using their priorities.
     */
    private record Entry<T>(T elem, double priority) implements
            Comparable<Entry<T>> {

        @Override
        public int compareTo(Entry<T> other) {
            return (int) Math.signum(priority - other.priority);
        }
    }
    ...
}
```

Overview of today

Sets

- Set Basics ✅
- Set Performance by Representation ✅
- Set Implementation (Live coding) ✅

Maps

- Map Basics ✅
- Map Implementation ✅
- Dynamic Priority Queues ✅

Metacognition

- Take 1 minute to write down a brief summary of what you have learned today

closing announcements to follow...

Sets

- Set Basics
- Set Performance by Representation
- Set Implementation (Live coding)

Maps

- Map Basics
- Map Implementation
- Dynamic Priority Queues

Announcements

- Coming soon: Ed post about Prelim 2