CS 310: Midterm Standings, Map From Set

Chris Kauffman

Week 9-2
## Midterm Scores

<table>
<thead>
<tr>
<th>Sec</th>
<th>F2015</th>
<th>F2014</th>
<th>R2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.</td>
<td>30.00</td>
<td>28.0</td>
<td>41.7</td>
</tr>
<tr>
<td>1stQ</td>
<td>64.00</td>
<td>68.0</td>
<td>65.0</td>
</tr>
<tr>
<td>Median</td>
<td>78.00</td>
<td>78.0</td>
<td>75.0</td>
</tr>
<tr>
<td>Mean</td>
<td>74.81</td>
<td>74.9</td>
<td>73.9</td>
</tr>
<tr>
<td>3rdQ</td>
<td>88.00</td>
<td>88.0</td>
<td>84.2</td>
</tr>
<tr>
<td>Max.</td>
<td>100.00</td>
<td>100.0</td>
<td>98.3</td>
</tr>
</tbody>
</table>
Graphical Distribution of Grades

F2015 Midterm Percentages
Bin width = 10

F2014 Midterm Percentages
Bin width = 10
Midterm Post-Mortem

Questions folks have on the midterm?
Feedback Results

- Posted
- Let’s take a look
Logistics

Reading

- Weiss Ch. 6 Java Library Classes/Interfaces
  - 6.7 Sets, 6.8 Maps
- Weiss Ch 18 Trees
  - Ch 18.4: - Iterator classes for traversals
- Weiss Ch 19 Binary Search Trees

Today

- Midterm Exams Back
- Midterm Feedback
- Map from Set
- General Trees and Traversals
Make a Map From a Set

Prototypes

```java
public interface SimpleSet<T> {
    boolean contains(T x);
    boolean add(T x);
    boolean remove(T x);
    T get(T x);
}
```

```java
public class MapFromSet<K, V>{
    // Trick 1: Use internal key/val class
    public static class KeyVal<K, V>{...}
    // Trick 2: Given a working Set class: use it!
    private SimpleSet< KeyVal<K,V> > theSet;

    // Implement these using theSet
    public MapFromSet();
    public void put(K key, V value);
    public void remove(K key);
    public boolean contains(K key);
    public V get(K key);
}
```

Exercise

Implement the put(), remove(), contains(), get() methods
The other Direction: Build a Set from a Map

Given a SimpleMap

- (key,value) pairs
- Each key is unique
- Insert value into a map according to its key
- Same key maps to same "place" in the data structure
- Supports `put(k,v)`, `remove(k)`, `contains(k)`

Build a SimpleSet

- Use an internal SimpleMap
- Implement SimpleSet methods
- `void add(T x)`
- `void remove(T x)`
- `T get(T x)`
- `boolean contains(T x)`

A Great Exam Question

- Tests your use of generics
- Illustrates abstraction skills
- Show you’re a proper software engineer
Java Does it

In Java: Map → Set

- `java.util.TreeMap` is a red-black tree
- `java.util.TreeSet` uses a `TreeMap`
- `java.util.HashMap` is a separate chained hash table
- `java.util.HashSet` uses a `HashMap`

In Weiss: Set → Map

- `weiss.util.TreeSet` is an AA-tree (another balanced tree)
- `weiss.util.TreeMap` uses `TreeSet`
- `weiss.util.HashMap` uses `HashSet`

Lesson

- Re-use when it makes sense
- Think hard about when it makes sense to re-use
Ordering

List property
There is a well defined ordering of first, next, last objects in the data structure

▶ A property of the Data Structure being used
▶ Wide ranging uses
▶ Supported structurally in LinkedList, ArrayList
▶ Not supported in hash tables (why not?)

Sorting property
There is a well defined ordering relation over all possible data of a particular type

▶ A property of the Data being stored
▶ "bigger than" "less than" "equal to" are well defined
▶ A data structure can try to mirror the data ordering structurally
▶ Useful for searching, walking through stored data in order
**Sorted Lists**

Definition is straight-forward

- "Smallest" things are structurally "first", "Biggest" last
- Ordering on elements (Comparable/Comparator)
- add/insert put elements in proper place

**Question:** For a sorted List $L$, what is the complexity of $L.insert(x)$ which preserves sorting?

$L$ is an `ArrayList`

- How long to find insertion location?
- How long to complete insertion?
- How long to traverse elements in order (e.g. for printing)?

$L$ is a `LinkedList`

- How long to find insertion location?
- How long to complete insertion?
- How long to traverse elements in order (e.g. for printing)?
Alternatives to the Linear Data Structures

Hash Tables
- Abandon list property
- Abandon sorting property
- $O(1)$ insertion/retrieval
- $O(N)$ traversal, not ordered

Trees
- Abandon list property
- Preserve sorting property
- $O(\log N)$ insertion/retrieval
- $O(N)$ traversal, ordered
- Commonly Binary Trees
- Other variants
Next the 2 weeks we’ll talk about roots
For simplicity, we’ll call them trees