CS 310: Maps and Sets

Chris Kauffman

Week 9-1

Logistics

Goals Today

- HW2 Discussion
- Maps and Sets

HW2 Discussion

- Milestones due Thu 7/6
- Discuss `AdditiveList`
- Iterator Implementation
- O(1) Undo/Redo

Reading from Weiss

- Today: Ch 6.7-9 Maps & Sets
- Upcoming: Trees
- Ch 18 Trees
- Ch 19 Binary Search Trees
- Weiss Ch. 7 Recursion

Operation Complexities (Speed)

- `add(x)`: put `x` in the DS
- `removeLast()`: get rid of "last" item
- `remove(x)`: take `x` out of DS
- `contains(x)`: is `x` in DS?

	add(x)	removeLast()	remove(x)	contains(x)
ArrayList	O(1)	O(1)	O(n)	O(n)
LinkedList	O(1)	O(1)	O(n)	O(n)
Hash Table	O(1)	X	O(1)	O(1)

This table is slightly misleading

- Careful of semantics of each operation
- Presence/lack of sorting property
- Set/Map distinctions
- What about space complexity of each?

Idea and Implementation

List

- List Idea: Ordered collection, accessible by numeric index: `l.get(i)`
- List Idea is formalized in Java's `interface List`
- `ArrayLists` and `LinkedLists` are both implementation of the List idea with different operational tradeoffs (describe)
- Could one implement a List with a hash table: `HashList`?

Set and Map

- Useful ideas: Set of unique items, Mapping of keys to values
- Can implement set or map with a variety of data structures
  - Arrays, Linked Lists, Hash Tables, Trees

Map and Set

Set: `HashSet` and `TreeSet`

- Collection of distinct objects
- Supports `add(x)`, `remove(x)`, `contains(x)`, sometimes `get(x)`
- `x` is either in the set or not in the set

Map: `HashMap` and `TreeMap`

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

Examples of Sets and Maps

A data type

```
class Student{
  String name;
  int gNumber;
}
```

A set of students

```
Contents
{Kyle, 1234}
{Stan, 4321}
{Eric, 2486}
{Kenny, 1313}
{Stan, 1357}
```

A map of IDs to students

```
Key      Value
1234 -> {Kyle, 1234}
4321 -> {Stan, 4321}
2486 -> {Eric, 2486}
1313 -> {Kenny, 1313}
1357 -> {Stan, 1357}
```

A map of Students to Majors

```
Key                  Value
{Kyle, 1234}   -> World Religions
{Stan, 4321}   -> Geology
{Eric, 2486}   -> Nutrition
{Kenny, 1313}  -> Mortuary Sciences
{Stan, 1357}   -> Genetics/Cloning
```

Questionable Sets and Maps

A set of majors?

```
Contents:
World Religions
Geology
Nutrition
Mortuary Sciences
Genetics/Cloning
```

A set of names?

```
Contents:
Kyle
Stan
Eric
Kenny
Stan
```

A map of IDs to names?

```
Key      Value
1234 -> Kyle
4321 -> Stan
2486 -> Eric
1313 -> Kenny
1357 -> Stan
```

A map of names to IDs?

```
Key       Value
Kyle  -> 1234
Stan  -> 4321
Eric  -> 2486
Kenny -> 1313
Stan  -> 1357
```

Array Analogy

Arrays and `ArrayList` are like a `Map` where

- Keys are integers: store at array index
- Values are the objects at those indices

A `Set` of `Integers` is naturally represented as an array of booleans

- Represent sets of `Integers` 0 to 10
- Use arrays of size 11
- The set $\{1, 8, 9\}$ is the array

```
boolean set1[] = new boolean[]{
  false,true,false,false,false,false,
  //    1
  false,false,true,true,false
  //          8    9
};
```

More efficient with `BitSet` if you're willing...

General Observations

Set

- A set must guarantee uniqueness of elements
- Typical approach is during `add(x)`, check `contains(x)` and don't add duplicates but there are other approaches
- Efficient implementation of `contains(x)` and `get(x)` becomes important for sets

Map

- The collection so keys is a set - each key must be unique
- `contains(k)`/`get(k)` important - make them efficient
- Collection of values is not unique
- Usually not efficient to look up whether a given value is present
- Collection of (key,value) pairs is unique due to keys being unique

General Implementations

How would you implement a `Set<T>`?

- Using an `ArrayList`?
- Using an `LinkedList`?
- Using a Hash Table?

How would you implement a `Map<K,V>`?

- Using an `ArrayList`?
- Using an `LinkedList`?
- Using a Hash Table?

General Solutions

Set from a Array or Linked List

- Guarantee items in list are unique by searching
- Could sort array for efficient lookup via binary search
- Still looking at $O(N)$ operations somewhere

Map from Array or Linked List

- Keep track of Pairs of (Key,Value)
- Lookup is based on only Key
- Can sort based on Key part only
- $O(N)$ operations somewhere

Set from Hash Table

- Good fit: $O(1)$ lookup via hash codes
- Keep load low and your in good shape

Map based on Hash Table

- Keep track of pairs of (Key,Value)
- Hash only the Key part
- $O(1)$ lookups if load is low

Have Set, Build Map

Q: If I have `Set`, how would I build `Map`?

Given: `SimpleSet`

- Collection of distinct objects
- Uniqueness determined by `equals()` method
- Operations `add(x)`, `get(x)`, `remove(x)`, `contains(x)`
- `SimpleSet` implementation may be based on arrays, hash tables, trees, linked lists... you don't know

Build: `SimpleMap`

- Set of (key,value) pairs
- Compare pairs only on whether their key is equal

```
(a,x) == (a,x)
(a,x) == (a,y)
(a,x) != (b,y)
(a,x) != (b,x)
```

- Use the set to ensure no redundant keys enter
- Implement `put(k,v)`, `get(k)`, `remove(k)`, `contains(k)`

Trick 1: Use an internal class

```
public class MapFromSet<K, V>{
  // Trick: Use a nested class
  // Class to carry around (key,val) pairs
  public static class KeyVal<K, V>{
    public K key; public V value;
    public KeyVal(K key, V value){
      this.key = key; this.value = value;
    }
    // Required for any set to work
    // Compare only based on key
    public boolean equals(Object o);
    // Required for HashSet to work right
    public int hashCode();
    // Required for TreeSet to work
    public int compareTo(KeyVal<K,V> kv);
  }
```

Trick 2: Use a set of the key/val pairs

Prototypes

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