CMPSCI 187: Programming With Data Structures

Lecture #35: Maps, Course Review
9 December 2011

Maps, Course Review

- The Map Interface
- Implementations of Maps
- Course Review: Programming Techniques
- Course Review: Data Structures
- Course Review: Analysis of Algorithms
- Coming Attractions in CMPSCI

The **`Map`** Interface

- A **map** is a set of key-value pairs, which we can also view as a set of keys (no duplicate keys are allowed), or a collection of values. It doesn't have its own iterator method but each of the three views will have one.
- We can make a `Map` unmodifiable by not supporting certain operations.
- The `AbstractMap` class gets everything you need from `entrySet`, plus `put` and the iterator's `remove` if the map is to be modifiable.

```
public interface Map<K,V> {
   public boolean containsKey (Object key);
   public boolean containsValue (Object value);
   public Set<Map.Entry<K,V>> entrySet( );
   public V get (Object key);
   public Set<K> keySet ( );
   public V put (K key, V value);
   public V remove (Object key);
   public Collection<V> values( );
```

Implementations of Maps

- The `TreeMap` class implements a map by keeping keys in what is effectively an ordered list, using red-black trees to carry out all the `Set` operations on the keys in O(log n) time. The iterator respects the order on keys.

- The `HashMap` class, as we saw before, keeps its keys in a hash table. (Essentially, it is a HashSet of entries (key-value pairs), where the hashcode function operates only on the key of the pair and keys may not be duplicated.) Its iterator runs over the hash table and so will take longer than O(n) to iterate through all the entries if the table is much larger than necessary.

- Why use a map? We can manipulate a large set of data by just manipulating the keys. And we can make multiple collections out of the same set of data while storing it only once, by attaching different sorts of keys to the same values.

Course Review: Programming Techniques

- What can we do with Java now that we didn't know how to do at the start of the term?

- I hope we are more comfortable with inheritance, with using multiple classes, with generics, with applying general solutions to specific problems, and with using the Java API to work with specific library classes without looking at their code.

- The size and scope of the programs we've written should have forced you into using a more sophisticated design process than you needed in CMPSCI 121. Solving a problem normally means not just getting one output from one input, but creating a tool that can be used on multiple problems. Such tools are harder to test and debug. I probably should have spent more time on the specifics of testing and debugging, and design with stubs and drivers.

Course Review: Data Structures

- The course was organized around a series of data structures: stacks, queues, lists (unordered and ordered), trees, binary search trees, heaps, priority queues, and sets.
- We applied a powerful methodology -- define the behavior we want with an API, define and analyze one or more implementations of each class' interface, and learn about the Java library versions of each.
- Step back and look at what you are doing in Project #7 -- you are using a data structure, prefix trees, that is not in the book (but similar enough to other trees to be comprehensible), to make a more efficient search through a large data set. You're also using a priority queue to organize answers to a search of a substantial space of words, a search that you could code with loops, with an explicit stack, or with recursion. Using the priority queue may mean designing a new class just for that purpose.

Course Review: Analysis of Algorithms

- The mathematical portion of this course is a “first draft” for CMPSCI 311. We recognize that different implementations of the same interface can vary widely in time efficiency.
- Big-O notation gives us a language to talk about the relative efficiency of different approaches to a given problem.
- We have solutions to the all-stars list problem of Project #7 that vary from less than a second to an hour and a half. Big-O analysis can’t tell us how long this particular problem will take, because it is a specific problem rather than a parametrized family, but it can guide us away from exponentially-long searches and toward techniques that we’ve seen to be efficient in other settings.
- We’ve talked about correctness, but so far lack the techniques to prove it.

Coming Attractions in CMPSCI

- CMPSCI 220: Programming Methodology. Using advanced Java features effectively, design patterns, refactoring.
- CMPSCI 230: Computer Systems Principles. Basics of architecture and operating systems. Concurrency, synchronization, I/O, networking, distributed services. “What goes on under the hood.”
- CMPSCI 240: Reasoning Under Uncertainty. Counting problems, probability theory, Bayesian reasoning (decision-making in uncertain situations), Markov chains, information theory.
- CMPSCI 250 (me!): Introduction to Computation. Logic, set theory, number theory, proof by induction, trees and searching, finite-state machines and regular languages, a bit of computability.