CSCI 230 Data Structures and Algorithms
Problem Set 3 - Maps, Hash Tables, and Sets
Jonathan Limpus

Assignment

This assignment is based on material from the course primary textbook, "Data Structures and Algorithms in Java" by Michael Goodrich, chapters:

- Chapter 10 Maps, Hash Tables, and Skip Lists

Problem 1. The use of null values in a map is problematic, as there is then no way to differentiate whether a null value returned by the call `get(k)` represents the legitimate value of an `entry(k,null)`, or designates that key `k` was not found. The `java.util.Map` interface includes method `boolean containsKey(k)`, that resolves any such ambiguity. Implement such a method for the `UnsortedTableMap` class.

Code: `UnsortedTableMap.java`

```java
public class UnsortedTableMap<K,V> extends AbstractMap<K,V> {
   ...
// easy way: use the methods defined in UnsortedTableMap
   boolean containsKey(K key) {
    // Call the method get, which returns null if there is no value is found
        if(get(key) == null) {
            return false;
        }
        else return true;
    }

// second way: Use java ArrayList and Object methods
    boolean containsKey(K key) {
        int size = table.size();
        for(int i = 0; i < size; i++) {
            if(table.get(i).equals(key))
                return true;
        }
        return false;
    }
   ...
}
```

Problem 2. What is the worst-case time for putting n entries in an initially empty hash table, with collisions resolved by chaining? What is the best case?

- The worst case time for putting n entries in an empty hash table would be $\mathcal{O}(n)$, which is assuming you're checking every element of the hash table.
- The best case time would be $\Omega(1)$, if the elements were always added to the front. This assumes that the items are always added to the front of the hash table.

Problem 3. Describe how a sorted list implemented as a doubly linked list could be used to implement the sorted map ADT.

Solution. A sorted map uses the following methods, as defined in the course textbook: **size(), isEmpty(), get(k), put(k,v), remove(k), keySet(), firstEntry(), lastEntry(), ceilingEntry(k), floorEntry(k), lowerEntry(k), higherEntry(k)** and **subMap**(k_1, k_2). A sorted map would use the following implementations Linked Positional List (a list implemented as a doubly linked list) methods:

- The sorted map **size();** would simply return the method **size();** from the Linked Positional List ADT.
- Again, the sorted map **isEmpty();** would inherit from the Linked Positional List **isEmpty();** directly.
- For **get(k)**, you would use either **before(k + 1)** or **after(k - 1)**
- For **put(k,v)**, you would use the method **set(k,v)**
- For **remove(k)**, you would use the Linked Positional List **remove(k)**
- For **keySet()**, you would iterate through the list and get the positions of each element using **after()** and add them to some Iterable object
- **firstEntry();** would be implemented using **first();** from the Linked Positional List class.
- Similarly, **lastEntry();** would use **last();**
- **subMap**(k_1, k_2): add the element at the first key using **get(k)** and **addFirst(k)**, then iterate through the list, adding values using Linked Positional List **addLast(v)** until you reach the final value.

Problem 4. What abstraction would you use to manage a database of friends' birthdays in order to support efficient queries such as "find all friends whose birthday is today" and "find the friend who will be the next to celebrate a birthday"?

Solution. Using a sorted map, and using dates as keys, you could quickly parse through the birthday. To find all "birthdays today", you could use the **subMap**(k_1, k_2) method using today's date as k_1 and k_2, and to find the "next birthday", you could use the **firstEntry()** method to find the soonest upcoming birthday.