Homework 5 Summer 2015
Cuckoo Hashing

Goal: To implement a hash table using an algorithm that guarantees at most 2 lookups are required to find something.

General: We learned in class that a problem with hash tables that use linear probing or external chaining was that the performance could degenerate from our expected time of O(1) to O(n). Cuckoo hashing is a table that has worst case performance of O(1).

The original paper for Cuckoo hashing is included in the homework assignment if you would like to read the entire paper. The basic goal of the cuckoo table is to require at most 2 comparisons to find something in the table. To accomplish this, we will need two backing arrays for tables and two different hash functions (Note this is slightly different from the original paper which had only one large table).

To insert into the table, we use the first hash function to compute an index and compress as normal. We then look at the first table, if the slot is empty, we just insert the new value. If it is not, we push out the value that is there (thus the name cuckoo where the bird pushes its young out of the nest). We then try to insert the pushed out item into table 2 using the second hash function. If something is there, we push out that item and then try to insert the pushed out item into table 1 again. This continues until we either succeed, or we do enough swaps that we think we are endlessly swapping out items. In that case we do a regrow.

A couple of things are slightly different about cuckoo hashing than the tables we talked about in class. One of them of course, is that we need two hash functions. The other is that those hash functions have to change on every regrow. You are free to come up with whatever technique you like, but a simple solution is to generate two random integers p1 and p2. Then just XOR those by the output of your hash function. Each time you regrow, just generate two new values and your hash is effectively changed. Warning - do not change p1 and p2 between regrows!

Implementation:

You will implement a CuckooMap that implements the Java.util.Map interface (See grading criteria for list of methods). You will also need a couple of private methods for sure like loadfactor() and regrow(). My class def looked like:

public class CuckooMap<K extends Hashable, V> implements Map<K, V> {

I have provided the Hashable interface, along with two classes that implement that interface. These classes also demonstrate how to cache the hash code so you don't have to compute it every time.

You will also need a Bucket class. In order to use a TreeSet (discussed later), your Bucket will need to implement the Comparable interface. Since Hashable's are comparable, just use the key to implement your compareTo function. My inner class def looked like:

private class Bucket<K extends Comparable, V> implements Map.Entry<K, V>, Comparable

Some of the Java api methods require you to return a Set. For this homework, you may use the java.util.TreeSet implementation for those methods. For the methods that require a Collection, you may use the java.util.ArrayList implementation.

There are other papers and resources on the internet that can help you if you still do not understand Cuckoo Hashing. You can get help from the instructor or TA's. As always, you may not refer to any implementations that are on the web.

Your constructor should set the default table size to 11, and the default maximum load factor to 0.5. Regrow should be at a minimum double the size of the table and add 1 to maintain odd size. Even better is to insure that the regrown table is always of a prime size to ensure optimal spread of keys over table.

Here are some optimal primes: 11, 23, 53, 97, 193, 389, 769, 1543, 3079, 6151, 12289, 24593, 49157, 98317, 196613.... You can store these in a table and pick the appropriate size based on how many times you have regrown. The homework tests will not require you to regrow beyond the largest value above.

Coding:
For this homework 10 points are allocated to code quality. The TA's have noted some sloppy coding practices in the previous homeworks. Points will be deducted for poor code standards (lack of comments, indentation, spacing, etc), failure to make helper methods private, improper encapsulation (exposure of implementation details outside the class), Magic Numbers, etc.

Turn-in:
CuckooMap.java

Grading Points:
Code Quality …... 10
size() …..... 05
isEmpty()..... 05
containsKey() ….. 05
containsValue() ….05
get() …..... 05
put() …..... 15
regrow() …..... 15
remove()..... 10
putAll() …..... 05
clear() …..... 05
keySet()..... 05
values()..... 05
entrySet() …..... 05