CMSC 123 Lab 7

Goal: This lab will teach you about hash tables, hash codes, and compression functions.

Copy the files to your home directory.

Implement a class called HashTableChained, a hash table with chaining. HashTableChained implements an interface called Dictionary, which defines the set of methods that a dictionary needs. Both files appear in the "dict" package.

The methods you will implement:

- insert() an entry (key + value) into a hash table,
- find() an entry with a specified key,
- remove() an entry with a specified key,
- return the size() of the hash table (in entries),
- and say whether the hash table isEmpty().
- There is also a makeEmpty() method, which removes every entry from a hash table, and
- two HashTableChained constructors. One constructor lets applications specify an estimate of the number of entries that will be stored in the hash table; the other uses a default size. Both constructors should create a hash table that uses a prime number of buckets. In the first constructor, shoot for a load factor between 0.5 and 1. In the second constructor, shoot for around 100 buckets. Descriptions of all the methods may be found in Dictionary.java and HashTableChained.java.

  Do not change Dictionary.java. Do not change any prototypes in HashTableChained.java, or throw any checked exceptions. Most of your solution should appear in HashTableChained.java, but other classes are permitted. You will probably want to use a linked list code of your choice. Note that even though the hash table is in the "dict" package, it can still use linked list code in a separate "list" package. There's no need to move the list code or the "list" package into the "dict" package, nor is it a good idea.

  Look up the hashCode method in the java.lang.Object API. Assume that the objects used as keys to your hash table have a hashCode() method that returns a "good" hash code between Integer.MIN_VALUE and Integer.MAX_VALUE (that is, between

-2147483648 and 2147483647). Your hash table should use a compression function, as described in lecture, to map each key's hash code to a bucket of the table. Your compression function should be computed by the compFunction() helper method in HashTableChained.java (which has "package" protection so we can test it independently; DO NOT CHANGE ITS PROTECTION). Your insert(), find(), and remove() should all use this compFunction() method.

The methods find() and remove() should return (and in the latter case, remove) an entry whose key is equals() to the parameter "key". Reference equality (==) is NOT required for a match.

Compression functions

Besides the lecture notes, compression functions are also covered in Section 9.2.4 of Goodrich and Tamassia. If you have an old edition (prior to the fifth), they make the erroneous claim that for a hash code i and an N-bucket hash table,

  h(i) = |ai + b| mod N

is "a more sophisticated compression function" than

  h(i) = |i| mod N.

Actually, the "more sophisticated" function causes _exactly_ the same
collisions as the less sophisticated compression function; it just shuffles the buckets to different indices. The better compression function (which they get right in the fifth edition) is

  h(i) = ((ai + b) mod p) mod N,

where p is a large prime that's substantially bigger than N. (You can replace the parentheses with absolute values if you like; it doesn't matter much.)

Checkoff

Run the HashTableChained.java

Bonus(2pts): Construct a chart of the distribution of keys. You can use xchart library (google it).