Programming Homework Assignment #5

Due Date: Thur., June 8, 11:55 PM

Upload the source files (.java files) with output (copied and pasted to the end of the main file) ZIPPED into ONE .zip file (submit under Week 9)

Problem: Write a Java program which uses updated versions of HashSC AND HashQP classes. Use the IpAddress class given in the HW5_JavaFile. USE the UPDATED LList.java class given in the Hash Table Files folder under Week 6 in Catalyst (how to use is indicated in the HW5-CodeFile). Note that the Comparator<E> interface is in java.util.*

Update the HashSC AND HashQP class instance methods given in the HW5-Code file (NOT in the Hash Table Files folder) (I'm calling "Code file" below):

- make HashSC and HashQP SUBCLASSES of the abstract HashTable class (given in the Code File)
- add 2 interface parameters to the constructor, one for a Hasher<E> and another for a Comparator<E>, and pass them to their corresponding parameters in the superclass' constructor
- override the **getEntry**() method, like remove(), but assigns to returnedItem if found (need to write WHOLE method!)
- CHANGE in the HashSC class the **insert**, **getEntry**, and **contains** so any comparisions each method MUST traverse a linked list using its iterator (see **remove** and **rehash** for examples) AND will use the compare method from the Comparator (this is similar to the BinarySearchTree's use of the compare method) (DON'T call a linked list's contains() !)
- change in the HashQP class, the **findPos** so any comparision with Object will use the compare method
- add Code to HashSC's **insert** and HashQP's **findPos** so they will increment the collisionCount (declared in the superclass) ONLY when a collision occurs (see answers to Lab Ex.8.2), and will update the longestCollisionPath when there is a longer linked list (in HashSC) OR the while loop in HashQP's findPos has iterations. In HashQP, make sure you reset the counters to 0 if before rehashing.
- change **myHash()** in HashSC AND HashQP so it doesn't call x.hashCode, but the Hasher<E> instance variable's **hash** method
- write a **displayTable**() instance method as described in the HW5_JavaCodeFile

Define the following classes using the Hasher<E> interface given in the "Code file" and the Comparator<E> library interface (in java.util):

- **class IpAddressValueHasher implements Hasher<IpAddress>** so when you override hash(IpAddress) it returns a Long object for the parameter's ipValue's call to hashCode
- **class IpAddressStringHasher implements Hasher<IpAddress>** so when you override hash(IpAddress) it returns an int for the parameter's dottedDecimal String using the SAME algorithm shown in **int Hash(String key)** in Lesson 8, p. 8.2 (NOT String's hashCode), or your own algorithm that manipulates each char

- **class IpAddressValueComparator implements Comparator<IpAddress>** so it overrides only the **compare** method, return the return value of the Long's compare or compareTo for the first parameter's ipValue and second parameter's ipValue
- **class IpAddressStringComparator implements Comparator<IpAddress>** so it overrides only the **compare** method, return the result of first parameter's dottedDecimal's compareTo (passing the second parameter's dottedDecimal)

Write main so it declares 2 HashTable<IpAddress> variables (they MUST be inside main). Assign to one of the HashTable variables a new **HashSC<>** passing a new IpAddressValueHasher and a new IpAddressValueComparator, and the other to a new **HashQP<>** passing a new IpAddressStringHasher and a new IpAddressStringComparator. In main, do the following (each bullet should be a method):

1. call a method (you write) to fill the a HashSC and HashQP (both are parameters in ONE method because they will refer to the SAME data) (BUT THERE WILL BE AN IpAddress return value). Open an input file the same way you did in Prog. HW#1 (using the openInputFile method, also given in the Code file UNCHANGED). If the file doesn't open, return null. If it opens:
   - read the file which has several sets of IpAddress, which has a String, one per line (read to the end of line, then trim()), create a new IpAddress with data from the line of input, **insert** the same IpAddress instance to each hash table. Read until the end of file (use the Scanner's hasNext() method). (Optional: display if it successfully inserted or not.) Close the file at the end of the method, then ...
   - at the end of the method, RETURN THE LAST IpAddress object inserted into the HashTables, or null if the file didn't open or none were read
2. if the file-reading method wasn't successful (returns null), display "Unable to read the file" and end the program.
3. call the *displayTable* method AND call the *displayStatistics* for your HashSC and then your HashQP table, (make sure you display a description of which table is being displayed first)
4. check if the IpAddress returned from the readFile method is contained in the HashSC table (calling **contains**() method), and if it is, call the *testHashTable* method (in the Code file, MUST ALSO CALL YOUR OWN test method) for your HashSC table, ALSO passing the IpAddress RETURNED from the readFile method (make sure you display a description of which table is being tested first) (IF contains() RETURNS FALSE, DISPLAY A MESSAGE)
5. do step 5. again, this time for the HashQP table
6. AGAIN, call the *displayTable* method for your HashSC and then HashQP table (again, make sure you display a description of which table is being displayed first)

Extra Credit Problem (due the last day of the quarter!): Change the HashQP class so it doesn't use quadratic probing, but a Random object for rehashing (hints will be given in a separate file upon request)