**CSC 364 Assignment - 5**

**Total points: 40**

**Implementation of a HashSet using Quadratic Probing**

**Files Provided**

- MyQuadraticHashSet,java (*to be modified*)

- MySet.java (*implemented by MyQuadraticHashSet*)

- MyHashSet.java (*required for testing*)

- TestHashSetsSmall.java (*testing program*)

- TestHashSets.java (*testing program*)

- TimeHashSets.java (*testing program*)

**Implementation**

MyQuadraticHashSet implements the MySet interface. Use open addressing with quadratic

probing for implementing the hash set functionality in the **MyQuadraticHashSet.java** file.

Implement the sections which are labelled TO DO in the given MyQuadraticHashSet.java file.

**Probing Function**

Use the following quadratic function to calculate the index for each probe:

private static int probeIndex(int hashCode, long probeCount, int tableLength) {

return (int)((hashCode % tableLength + tableLength + probeCount \* probeCount) % tableLength);

}

Call the above method with *probeCount* = 0, 1, 2, 3, … for the probe attempts in your code.

**Class Constructor**

The MyQuadraticHashSet constructor requires two parameters: the load threshold, and an array

of prime numbers that should be used as table sizes. Proceed to the next value in the array each

time you need to resize and rehash the table. This array is provided by the test programs. The

first value in the array is 17, indicating that the table will initially have length 17. Your class will

be tested with load thresholds of 0.1 and 0.5.

**The Table Object**

The table is an Object[]array. You may be required to use casts of the form(E)in order to

get your code to compile. Use the @SuppressWarnings("unchecked") as necessary to

get your code to compile without “unchecked cast” warnings.

**Deleting Elements**

Removing elements in open addressing is slightly tricky. A naive approach is to replace removed

elements with the value *null*. However, this may short-circuit later probing sequences, yielding

incorrect results. Instead, I suggest that you declare a data field called REMOVED:

private final static Object REMOVED = new Object();

- Every time an element is removed from the table, replace it with REMOVED.

- When searching for an element, continue probing if a probe yields REMOVED

- When adding an element, replace an instance of REMOVED.

**Resizing Table**

Resize and rehash the table whenever the number of elements in the table plus the number of

occurrences of REMOVED exceeds thresholdSize.

**The iterator() method:**

Consequently, you do not need to implement an Iterator class for this assignment. However,

since your class will implement MySet<E>, which in turn extends

java.lang.Iterable<E>, you will need to have an iterator()method. Given that

none of the tests or timings use an iterator, it is all right to leave the iterator unimplemented.

**Testing Your Hash Set**

- **TestHashSetsSmall** will test MyQuadraticHashSet on a really small test case.

- **TestHashSets** will test it on larger cases, comparing its results on the add, contains, and

remove operations to the results of performing the same operations on a java.util.HashSet.

- **TimeHashSets** to compare its runtime performance to that of java.util.HashSet and

MyHashSet (the textbook’s hash set class).

**Sample Results:**

Here is a sample output from the TimeHashSets test program:

Each set will be timed on 2000000 add operations, 2000000 contains

operations, and 2000000 remove operations.

Timing java.util.TreeSet

Runtime: 3.484 seconds

Timing java.util.HashSet

Runtime: 0.766 seconds

Timing MyHashSet from textbook

Runtime: 0.953 seconds

Timing MyQuadraticHashSet with load threshold = 0.10

Runtime: 1.176 seconds

Timing MyQuadraticHashSet with load threshold = 0.50

Runtime: 0.681 seconds

Your runtimes do not need to match, but reexamine your code if your runtimes are drastically

worse.

**Submission:** Submit your modified and commented **MyQuadraticHashSet.java** file.