Assignment 3

Assignment 3: Hash Tables

CS102: Data Structures, Fall 2013
Eric Koskinen and Daniel Schwartz-Narbonne
New York University

Due: 4:55PM (16:55:00), Friday October 25th

Context

Your friend wants to extend their mobile app to be able to store arbitrary sets of facts about movies. For example, the app should be able to associate movie reviewers with the reviews they gave for the movie; associate days with the amount of money the movie made on that day, or associate words with the number of times they appeared in a movie script.

Having studied computer science, you realize that the obvious solution is a hash table. Since the map may need to store many different kinds of pairs of data, it needs to be generic. You roll a die and decide to use a chaining (i.e. linked list) based hash-table.

Scope

The focus of this assignment is a little different from the previous two. We have done most of the coding for you. Your job is to investigate how to design an effective and efficient hash-table. In particular, you will investigate the effect of different hash codes and load factors on the time it takes a hash-table to do some common tasks, and write a report on how to design good hash-tables.

Task 1: Write tests for the Hash-table implementation

Write tests for all functions exposed through the `Map<K,V>` interface. You are encouraged to write other tests, but our automatic grading framework will be expecting tests that use the `Map<K,V>` interface, and may not work if your tests do not follow this format, so you should only submit tests that follow this format.

As with all assignments, you will use the test-driven design approach. There is also a suite of testing tools that you will need to extend for this assignment.

- `TestHarness.java` -- A TestHarness class which provides the infrastructure for the testing suite. You do not need to modify this class.
- `Test1.java` -- An example of the kind of tests you should be creating.
  - Remember to focus on the Map<K,V> interface
  - Hint: this includes checking that the statistics were correctly calculated
- `Test.java` -- The main test function which registers individual tests and then invokes all of them. You will need to **extend the code** in `main` in this class and register any tests that you create.

As with the previous assignment, we will be developing some incorrect implementations, and running your tests cases on them. One of the jobs of your test suite is to try to catch all of our broken implementations.

Task 2: Finish the implementation of the HashTable class

We have implemented almost all of this class for you. All you need to do is

1. Implement the `remove(K key)` function. If an element with the key exists, remove it and return true; if no such element exists, return false and leave the table unchanged. Throw an exception if called with invalid paramaters.
2. Implement the `calculateStats()` function. It should calculate the following statistics and store them into a new HashTableStats object, which it should then return
    a. `loadFactor` is the ratio between the number of elements and the total size of the table: `numElements` / `table.length`
    b. `meanChainLength` is the average length of the linked lists in the table
    c. `stdevChainLength` is the standard deviation of the length of the linked lists in the table. You can calculate it using the formula:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}, \quad \text{where} \quad \mu = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

    Where x_i is the length of the list in the i^{th} table slot,and μ is the `meanChainLength` you calculated in part b.
    d. `longestChainLength` is the length of the longest linked list in the table.
    e. You are encouraged to calculate other statistics about the table which will help you design a better hash function. These should be printed out to the console. DO NOT ADD EXTRA FIELDS TO THE HashTableStats class.

Task 3: Investigate the effect of hashcode and load factor on hash table performance

Task 3a: **Finish the implementation of the SpeedTest class**

This class will be the major class you will use to investigate hash-table performance. It does two things:
1. Reads a set of words from a book or movie script, and stores the number of times each word occurs into a hashtable. For example, given a book with the words
"Who knows? The Shadow knows!"
-WHO should be in the hashtable with the value 1
-KNOWS should be in the hashtable with the value 2
-THE should be in the hashtable with the value 1
-SHADOW should be in the hashtable with the value 1
Looking up any other word should give the result null. We have implemented most of the code to do this, including the code to process the book into (uppercased) words. **You need to implement the code to take these words and store their count into the hash-table.**
2. Reads a set of words from the scrabble dictionary, and determines which word is associated with the highest value in the hash-table (i.e. occurred most frequently in the book/screenplay). Again, we have written most of the code for this - **you only need to write the inner loop that does the lookup and keeps track of the highest valued result.**

Task 3b: Write a new String hashing function

We have provided you with several string hashing functions.
- `StringHasher1` is based on the Java `.hashcode()` function
- `StringHasher2` & 3 are hash functions that we wrote, which as you will see do not lead to ideal hash-table performance
- **Your job is to write** `StringHasher4`**. Your goal is to write a hash function that makes the**

code in SpeedTest (described below) run as fast as possible.

Task 3c: Use the SpeedTest class to investigate the effect of different hash-codes and load factors on hash table performance

The SpeedTest class comes builtin with code which measures the time it takes to do various operations, using the Date class. Since the runtime for any individual run of a piece of code may vary from execution to execution due to interference from other programs running on the computer, the most reliable way to measure the runtime of a piece of code is to execute it inside a loop and measure the time for number of successive executions.

Your job is to use this class to determine the effect of different hash-codes and load factors. Write a report in word or pdf format which analyzes (using a combination of graphs and text) using the book`WarAndPeace.txt`.

1. For each of the four hashing functions (including your own)
2. For load factors: 8,4,2,1,½,¼,⅛

- The runtime for the two tasks
    - Loading the hash table.
    - Looking up words in the hash table
- The correlation between the runtime and various statistics such as the longest chain length, the standard deviation of chain length, etc
- Discuss how your analysis lead to your choice of hash-function for `StringHasher4`.

Compilation and Execution

Your assignment will be automatically compiled as follows:

```
javac *.java
```

Note that the Java compiler will automatically compile other classes that are inherited. Finally, your code will be executed as:

```
java Test
```

Note that your implementation will be built and executed against your test cases, as well as our test cases. It is to your benefit to devise as many test cases as possible!

Submission

Please submit all .java and .txt files, as well as your report (whose filename should include your netid and name), to NYUClasses.

In particular, you should have modified the following files (and no others)

- `HashTable.java`
- `SpeedTest.java`
- `StringHasher4.java`
- `Test[2-n].java`
- `Test.java`
- `YourReport.doc / YourReport.pdf`

Grading

Your assignment will be evaluated based on the following:

- Sufficiency of the tests you design (33%)
- Your implementation, including clarity and organization of the code, and sufficiency of comments(33%)
- The quality of the analysis in your report (34%)

We look forward to your submission!
-- CS102 Course Instructors

For **bonus points (mandatory for honours students)**, you may

- Implement extra statistics which give you a better view into what is going on in the hash-table
- Redo your analysis using “RomeoJuliet.txt”