CS249 Project 5
Hash Tables

Overview

This project requires the completion of the following tasks:

- Write hash sets and hash tables
- Resolve collisions using double hashing and chaining
- Apply hash sets and hash tables to example problems

Unit Testing

This project requires that for each part you must run the included unit tests to validate and verify the correctness and integrity of your programs, as well as to drive a test-driven development approach. Remember that test-driven development means that test and compilation errors are not necessarily a bad thing; you should use them as an indication of you what code you need to create or fix next.

Start by downloading the interface and unit tests files provided for this assignment and setting-up JUnit for your specific Java development setup. Below are some quick tips for getting started running JUnit tests in several common development setups.

After you turn in your assignment you will be graded using these tests, as well as additional unit tests, so make sure you are thorough in your implementation.

IntelliJ:

- Create a new project and add the interface and test files to the *src* directory
- Open up one of the test files from within IntelliJ and click on one of the lines underlined with an error
- Press ***alt+enter*** (***option+enter*** on mac) and select the option to import the JUnit library into your project
- You should now be able to run each of the tests (***right-click*** the file and choose *run*), or all of the tests (***right-click*** on the project and choose to run all tests) once you create the classes required for this project

Eclipse

- Create a new project and add the interface and test files to the project *src* directory (you might need to ***right-click*** and refresh the *src* icon in Eclipse)
- ***Right-click*** on the project in the Package Explorer panel, go to ***Build Path->Add Libraries***, and select JUnit
- You should now be able to run the unit tests by ***right-clicking*** on the project or test files under Project Explorer, or by going to the ***run->run as*** menu

Command Line

For a command line development environment, you will need to perform several steps:

- Download the required JUnit JAR files to your project directory
- Add the JAR files to your project class path when compiling and running
- Create your own driver class (a class with a main method) to run the tests and print out the results

Hash Sets

Sets are data structure designed to hold non-repeated items in no particular order. They are often used to determine membership. In this case, we are going to use a hash set to determine if a particular word is or is not included in the list of Scrabble words quickly. Being that no order is considered, this ADT has complete freedom to optimize for speed of retrieval. Here we will build a hash set based on your textbook or online resources.

1. ***Implement a HashSet class satisfying the interface provided in ISet.java.***
   - Remember to provide a constructor with the intended size of the set.
   - Use the hashCode method of whatever object you are passed to determine that element's place in the hash array.
     i. You will need to use Math.modFloor to determine an index from the hash code, as the hashCode result could be negative.
   - Use a double hashing strategy to resolve collisions.
     i. As your double hashing function, convert your hash into a string and find the hash value of that string.
   - Remember, in order for strings to properly match you must compare your hashed objects using the .equals() method rather than using the == operator.
   - Ensure you can pass all unit tests.
2. ***Profile your HashSet***
   - Build multiple HashSet and add 100,000, 200,000 and 300,000 entries to each respectively. Time how long it takes to look up 10 times that many entries after they are created.
   - Set their capacities to be double that amount. Build multiple HashSet and add 100,000, 200,000 and 300,000 entries to each respectively. Time how long it takes to look up 10 times that many entries after they are created. Set their capacities to be 300,110 for all sets.
   - Use System.nanoTime() to determine how long the lookups take.
   - Include these numbers in your report and explain the behaviors as the number of entries increases.
3. ***Write a program that uses your HashSet class to validate scrabble words.***
   - Determine if the following words are scrabble words by adding code to your main method.
     i. Queue

ii. Navient
iii. Aa
iv. Possum
v. Phoney
vi. Bb
vii. Werd
viii. Titi

Hash Maps

Hash tables are similar to hash sets, but rather than merely storing the hashed item, we can associate a key with some unrelated element. This is known as a Map, or an Associative Array. Hash maps function in the same way as hash sets, but rather than storing the hashed element alone, they store a hashed element as a key alongside some other element. These can be used to create an extremely efficient lookup table between keys and associated elements even when there are an unlimited number of possibilities for the key.

4. *Implement a HashMap class satisfying the interface provided in IMap.java.*

- Remember to provide a constructor with the intended size of the map.
- Use a linked list strategy to resolve collisions
  i. Another interface IMap.IMapPair is provided for this purpose. Tests will use this interface to verify your HashMap's internal structure.
- Use the hashCode method of whatever object you are passed to determine that key's place in the hash array.
  i. You will need to use Math.modFloor to determine an index from the hash code, as the hashCode result could be negative.
- Remember, in order for strings to properly match you must compare your hashed objects using the .equals() method rather than using the == operator.
- Ensure you can pass all unit tests.

5. *Profile your HashMap*

- Build multiple HashMap and put 100,000, 200,000 and 300,000 entries to each respectively. Time how long it takes to look up 10 times that many entries after they are created. Set their capacities to be the number of entries.
- Build multiple HashSet and add 100,000, 200,000 and 300,000 entries to each respectively. Time how long it takes to look up 10 times that many entries after they are created. Set their capacities to be 300,110 for all sets.
- Use System.nanoTime() to determine how long the lookups take.
- Include these numbers in your report and explain the behaviors as the number of entries increases. Also, compare these results to the profiling of the HashSets from earlier.

Write-up and Submission

- For each part of the project, describe your approach, solutions, describe any difficulties that you encountered, and detail the efficiency of all non-test methods in your code using Big-O notation. Include any requested output and screenshots for each part.
- Add all of your *.java* files and your write-up document to a .zip or .tar.gz archive and submit it on Bb Learn. Make sure all of your java files are in the root of the zip file, rather than inside additional folders.
- You **must** write and submit your own code. You must also clearly identify and online or text sources that you used as references, any collaborators with which you discussed the project (or lack thereof), and how the source was beneficial.

Grading

This project is worth 100 points.

- 10 Points – Design and code quality
    - Good object-oriented design, consistent comments, white space, indentation, etc.
- 60 Points – Implementation quality
    - Demonstrate that your solution is complete and accurate based on unit tests.
    - 10 points – HashSet basics
    - 10 points – Separate chaining for collisions
    - 10 points – Scrabble Dictionary
    - 10 points – HashTable basics
    - 10 points – Double hashing for collisions
    - 10 points – Geocaching application
- 30 Points – Writeup and Submission