CMSC-401 Algorithm Analysis with Advanced Data Structures

Homework #3

Lecturer: Hong-Sheng Zhou *Due: Friday April 22nd*

Problem 3-1 (Hash Tables.) 10%

In this assignment, you will be implementing a hash table and investigate the performance of this very useful data structure. We will use various tools such as *linked lists, hash tables, hash functions, and collision resolution with chaining* during this exercise.

This assignment will likely take a *significant* amount of time to complete (not one night), please allocate your time appropriately. You should name your main source file `cmsc401.java` and place it in a zip file called `SURNAME_GIVENNAME.zip` and submit it to blackboard by April 22, 2016.

1 Problem Statement

You are designing a new service for hitch-hikers that will help estimate how far they are from their destination. When a hitch-hiker sends his source and destination city, the service will calculate the total distance he will have to travel. Hitch-hikers can also use this service to look up their current coordinates.

1.1 The Service

For this assignment, you will need to implement the following components,

1. **Storage.** For performance reasons, you will store your city and coordinates data in a hash table.
2. **Retrieval.** The search engine will receive a *city* (key) and retrieve from the hash table the corresponding *latitude* and *longitude* (value) of that city.
3. **Calculator.** Based on the coordinates of two cities (source and destination), this calculator must provide the distance between the two points in **kilometers**.

1.2 Input Dataset

The dataset you will be provided for this assignment is a CSV file called **cities.csv**. It includes three columns: *city_name, latitude, longitude* in the format provided in an example below. The CSV file is available on blackboard. For testing information, see the example on the last page.

```
Kabul,34.5166667,69.1833344
Kandahar,31.6100000,65.6999969
Mazar-e Sharif,36.7069444,67.1122208
Herat,34.3400000,62.1899986
Jalalabad,34.4200000,70.4499969
```

2 Assignment Requirements

Your code must provide the following functionality:

1. **Insert** new values into the hash table.
2. Your program should **retrieve** the coordinates (value), given a city (key).
3. Your program should **read a CSV** file and insert all entries into the hash table using each *unique* city as a key. This file, called *cities.csv* will be passed in as `args[0]` to the main method of your program.
4. Keys are given as strings, so they should be **preprocessed by a hash function** as to ensure non-negative integers. You may use the `Object.hashCode()` method for this.
5. **Hash** the preprocessed keys (cities) using either a division or multiplication based hash function.
6. Collisions (when two keys hash to the same set/bucket) should be resolved by **separate-chaining**.
7. Your program should continuously loop and process the following commands from standard input (`System.in`): `retrieve arg; distance arg1 arg2; stop`. See the note section for details about the commands.
8. Upon reading the `retrieve arg` command, your program should **print** the coordinates of the city (`arg`).
9. Upon reading the `distance arg1 arg2` command, your program must calculate the distance between two cities (`arg1 arg2`) using the *haversine* formula.

$$d = R \cdot 2 * \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_1 - \phi_2}{2}\right) + \cos(\phi_1)\ \cos(\phi_2)\ \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{1}$$

   with R being the radius of the earth (6373 km), ϕ_1 and ϕ_2 as the latitudes of the points 1 and 2 respectivley, and λ_1 and λ_2 are the longitudes of points 1 and 2 respectively.[1]

10. Upon reading the `stop` command, the command loop should **terminate** and your program should **print** the average length of the linked lists (i.e, the average number of collisions in each set/bucket)

$$\mu_l = \frac{\sum_{i=0}^{B} \text{length}(l_i)}{B} \tag{2}$$

    with μ_l being the average length of the linked lists, B as the number of buckets in your hash table, and length(l_i) as the size of the linked list l at bucket index i.

[1] `https://en.wikipedia.org/wiki/Haversine_formula`

3 Grading

1. The *correct* implementation (60%)
2. The *successful* execution using our test file (40%)

4 Notes

1. The number of keys will be static, so you do not need to dynamically adjust your hash table (although that is a useful quality and we invite you to do so).
2. We will not try to "break" your program with fuzzy input; error handling never hurts though.
3. The exact method your program will be tested, how commands are passed, and when to print a value is demonstrated in the following example. In this example, `student@learning$` indicates a console/command line in the directory of your unzipped source file(s).

```
student@learning$ javac *.java
student@learning$ java cmsc401 cities.csv
  distance Boston New York City
  305.01145030334675
  retrieve Dallas
  32.7830556 -96.8066635
  stop
  1
student@learning$
```

   In this example, the value of μ_l is 1 indicating that this is a perfect hash table; this is the ideal case and it is unlikely that you will also get 1. Also, notice that "New York City" is three words, you will have to catch these types of cases. The indentation of part of this example is to emphasize that we are interacting with the java runtime environment (JRE) via `System.in` and `System.out`.

4. You are expected to implement your own linked lists and hash tables. **Do not use the built in java libraries for these data structures (i.e., hashmap, linkedlist, etc.)**. Please cite any references that you use in your implementation.
5. If you have any questions about submissions, clarity, or need help, your TAs are available to help in person or via email at {`vealetm; trinpm`}`@vcu.edu`.