CS 241: Data Structures II - Fall 2011

Programming Assignment 3

This is an individual assignment. In this assignment we will put to practice our knowledge of hashing. This assignment will be more about experimentation than developing an application. You will perform an experimental assessment of the efficiency and efficacy of different hashing techniques.

You are to implement **4** different hash tables, with different hashing functions. All of them should implement the following interface:

```
public interface HashTable<K,V> {

  public void add(K key, V value);

  public V remove(K key);

  public V lookup(K key);

  public void printReport();

}
```

The `printReport()` method should print to the console the following statistics:

- The **Load Fator**, that is, the ratio of used to total number of buckets.
- The **longest chain** in the table, that is, the **maximum number of collisions** for a particular bucket.
- The **Density Factor**, that is, the ratio of elements stored elements to total number of buckets.
- The **Chanining Factor**, that is, the average length of any chain in the table.

The values stored will be pairs of string of length 4 and int values. You are to create a procedure as part of your assignments that will generate these pairs by implementing the following interface:

```
public interface KeyValueGenerator {

  public void initialize();

  public String getNextKey();

  public int getNextValue();

}
```

The `initialize()` method initializes the internal state of the generator to some random string of 4 letters (always lower case), which will be the initial key. After the initial key is given out with the method `giveNextKey()`, subsequent keys are random 4 letter strings different from the previously generated. To achieve this, simply use the use the transformation function described below to obtain a unique integet ID form the string and store it in an array, and use it to check that the string hasn't been used before in the current run.

Also, make sure to re-seed your random number generator on every run. The method `getNextValue()` always returns a random int between 1 and 100.

The 4 types of hashing functions you are to implement are:

- **Additive Hashing:** the hash is determined by adding the numeric value of every letter, mod the size of the table. The numeric value is obtained with the obvious assignment to every letter: a = 0, b = 1, c = 2, ... , y = 24, z = 25, and then multiplying by 26^p, where p is the position of the letter in the string, starting from 0 for the rightmost, all the way up to 7. For example, in the string `eiwz`, the value of the z is 26 x 26^0 = 26, whereas the value of e is 5 x 26^3 = 87880. So, if d_i is the numeric value of the ith letter in the string, then the hashing function is:

  $$hash(s) = sum(d_i) \bmod SIZE$$

- **Multiplicative Hashing:** same as additive, but using multiplication instead of addition.
- **Rotation-Additive Hashing:** same as additive hashing, except the numeric value of every letter is **shifted 4 positions to the left** before it is added to the current partial hash (which is also rotated). For example, if the numeric value of the letter is 87880, then shifting it for positions to the left would yield 08788, where every digit is shifted to the left 4 times. You are to implement this rotation operation.
- **Rotation-Multiplicative Hashing:** same as rotation-additive, except using multiplication instead of addition.

The assignment will have 2 parts:

No Resizing/Rehashing

You are to implement all 4 techniques with tables that provide no resizing mechanism, and you will set up the following experiments:

1. Create tables of each type of size 100 and use the automated key-value generator to fill them with 25 , 50, 75, 100, 125, and 150 elements. Each experiment must be repeated 50 times, for every table, for every number of elements. You will collect the results using the reporting method, and compute the average values for each experiment.
2. Same experiment as before, except with table sizes of 1000, and 250, 500, 750, 1000, 1250, and 1500 elements.
3. Same experiment, except with table sizes of 10000, and 2500, 5000, 7500, 10000, 12500, and 15000 elements.

Resizing/Rehashing

You will implement a second version of the tables, but now with resizing/rehashing. Your criteria for resizing the tables will be **surpassing a load factor of .75 or a longest chain of 5**.

You will perform the same experiments as in the previous section and gather the data. Initially I had designed extra experiments here that dealt with computing timing, but that has been removed in favor of the other things. So no computation of timing will be needed as I had hinted in class.

Report

After collecting all the data you are to write an thorough report discussing your results. Look at every statistics and draw conclusions from what you observe to determine which techniques are best on what circumstances. What is a good values for each factor? Is it good to have a high or low load factor? How about density? Why? Why not? Taking into account how efficient the table is in terms of both space and time to lookup an element.

Deliverables

Your submission, to be made by email **before November 15, 2011 at 12:00am**, should contain the following items:

- **Source Code:** You will of course provide source code for your solution (no binaries), which should be **properly documented** with Javadoc. You must also submit HTML documentation generated from Javadoc. The code should be delivered in a **.zip** file, or a compressed **.tar** file (**.tgz**). You should also include a README file with any information pertinent to the execution of your program (e.g., command line options, etc.). Your source code should also contain functions to easily duplicate all your experiments.
- **Documentation:** You will submit a project report, that will contain at least: a description of the project, approach, experimental data, discussion of experiments, and conclusions/lessons learned.

This project will be worth **50 points**. You may have noticed this is a non-trivial project, so allocate time and resources appropriately, and do not hesitate to aproach me with any questions/doubts you might have, or if you think I can help you in any way.

Edwin Rodríguez. Last updated: November 8, 2011