CSC172 LAB 17

HASHING

1 Introduction

The labs in CSC172 will follow a pair programming paradigm. Every student is encouraged (but not strictly required) to have a lab partner. Labs will typically have an even number of components. The two partners in a pair programming environment take turns at the keyboard. This paradigm facilitates code improvement through collaborative efforts, and exercises the programmers cognitive ability to understand and discuss concepts fundamental to computer programming. The use of pair programming is optional in CSC172. It is not a requirement. You can learn more about the pair programming paradigm, its history, methods, practical benefits, philosophical underpinnings, and scientific validation at http://en.wikipedia.org/wiki/Pair_programming.

Every student must hand in their own work, but every student must list the name of their lab partner if any on all labs.

This lab has six parts. You and your partner(s) should switch off typing each part, as explained by your lab TA. As one person types the lab, the other should be watching over the code and offering suggestions. Each part should be in addition to the previous parts, so do not erase any previous work when you switch.

The textbook should present examples of the code necessary to complete this lab. However, collaboration is allowed. You and your lab partner may discuss the lab with other pairs in the lab. It is acceptable to write code on the white board for the benefit of other lab pairs, but you are not allowed to electronically copy and/or transfer files between groups.

2 Set Operations

The Hash table ADT supports various set operations. In this lab, you will implement basic hash table data structures.

1. Hash Function

   Begin this lab implementing the “good” hash function discussed in section 5.2 of you textbook in order to generate a hash key from any String.

2. Implement a simple hash table class that stores its data in an array of strings. Use open addressing with linear probing to resolve collisions. Implement two methods: insert and print. The print method should print all the (non null) strings in the table. Your insert method should

add the given string to the table if it is not already present. If asked to add a string already in the table, your insert method should do nothing (you can only have one of each string in the table). Test this by adding five names to a hash table of size 13. Add the names, then print out the items in the table.

3. Your hash table should have getter methods for its capacity, the number of unique items contained, and load factor.
4. Use the load factor modify your insert method to detect when the load factor exceeds 50%. When this happens, you should expand the array to double its original length and "rehash" the strings into the new array.
5. Test your hash table by generating ten paragraphs of *Lorem Ipsum* text (http://www.lipsum.com/) copy this text into a text file. Have your program read the file and enter the words into your hash table. Start with a hash table of size 13. At the end your code should print out the unique words, the number of unique words and the total count of words read in, and the final size of your hash table.

3 Hand In

Hand in the source code from this lab at the appropriate location on the BlackBoard system at my.rochester.edu. You should hand in a single zip file (compressed archive) containing your source code, README, and OUTPUT files, as described below.

1. A plain text file named README that includes your contact information, your partner's name, a brief explanation of the lab (A one paragraph synopsis. Include information identifying what class and lab number your files represent.), and one sentence explaining the contents of all the other files you hand in.
2. Java source code file(s) representing the work accomplished for this lab. All source code files should contain author and partner identification in the comments at the top of the file.
3. A plain text file named OUTPUT that includes author information at the beginning and shows the compile and run steps of your code. The best way to generate this file is to cut and paste from the command line.

4 Grading

90% Functionality

- 10% Hash function
- 30% Hash table class with insert and print methods
- 10% Size, capacity, and load factor methods
- 20% Resizing hash table when load factor is too high
- 20% Testing lorem ipsum file

10% README and OUTPUT files