CSCI 240 -- PA 7

Hashing

Feel free to discuss and help each other out but does not imply that you can give away your code or your answers! Make sure to read all instructions before attempting this lab.

You can work with a lab partner and each one must submit the same PDF file (include both names in the submission file). Each person must include a brief statement about your contribution to this assignment.

You must use an appropriate provided template from Canvas and output "Author: Your Name(s)" for all your programs. If you are modifying an existing program, use "Modified by: Your Name(s)".

Exercise 1 – Use C++ language STL **map** or Java language **HashMap** to store the following integer keys: 13 21 5 37 15 (use the reverse of the key as a string for the value part so first entry would be <13, "31">). Perform the following operations to make sure it is working properly: search for 10 and 21, remove 20, 37, and then search for 37. If an item is found, print the key and its value.

Input data file ***small1k.txt***, containing a list of 1,000 integer values and insert all the pairs <int, reverse key as string> to a new hash map. Collect the time it took to insert 1,000 pairs of values to the hash map and output the time to the screen.

Input data file ***large100k.txt***, containing a list of 100,000 integer values and insert all the pairs <int, reverse key as string> to another new hash map. Collect the time it took to insert 100,000 pairs of values to the hash map and output the time to the screen.

Exercise 2 – Put together the C++ **HashMap** in the book (Chain Hashing) or Java **ChainHashing** (Java book) with N = 11 and test it out with the same data and test cases from exercise 1. You might want to come up with all relevant test cases to confirm that C++ HashMap or Java ChainHashing is working correctly. Collect the times for the 2 data files like exercise 1. They should be like the times in exercise 1 and you should confirm that is the case.

Exercise 3 – Perform P-9.9 (pages 420-421 of C++ book). Use 0 (not a good value), 37, 40, and 41 for parameter **a** with polynomial hash codes or a shift of 0 (not a good value), 1, 5, and 13 for cyclic shift hash codes (only need to pick one method, either polynomial hash codes or cyclic shift hash codes). Test it on the text file, US Declaration of Independence, and you can assume that words are separated by white space characters. Output number of words in the file and the number of collisions for each

case. Make sure the results are reasonable (one of the 4 values should give a much higher number of collisions).

Question 1: Do collected times for exercises 1 and 2 make sense? Explain why or why not.

Question 2: Why do we need to use compression functions? What are some common compression functions?

You can earn EC for only one of the options below:

Extra Credit option 1 (2 points): Collect data for second hash code for exercise 3.

Extra Credit option 2 (much more extra work, 4 points): Create a program to collect some data about chain hashing that you worked on in exercise 2. You would be able to enter the name of the input data file and a load factor. Each input data file will have N records and the first value in the file will tell you how many records are in the file. You would need to use N and the load factor to determine the initial size of the hash table. The first value of each record will be the key (county/state code as integer type) and remaining items on each record will be the value (population and county/state and you can treat it as one string). After all the entries are inserted to the table, print the **table size, average number of probes, and maximum number of probes for the worst case**. It would take at least one probe for each insertion (checking initial location). Therefore, it would be two probes if a second location is examined.

After confirming that your implementation works reasonably well against the small input file (popSmall.txt), run it against the larger input file (popLarge.txt) and collect some required data. You must try the load factors of 0.25, 0.5, and 0.75 against the large input file (3 different runs). Make sure that your hash table size is determined correctly according to the load factor by using the following formula: N / LF + 1. For example, if a file has 5 values and for a test at load factor 0.5, the table size is 11 (5 / 0.5 + 1).

Fill out and turn in the PA submission file for this assignment (save as PDF format).