Assignment 4: Hashtables

Due: 10/26/2011, 11:59PM

1 The task

You will design a `Hashtable` class for `std::string` objects that is parameterized by a hashing function and a table size (number of buckets). The best way to parameterize the hashing function is by passing a Hashing functor as a template argument. The table size should be specified in the constructor.

Your program should then open the dictionary file (which can be downloaded from blackboard), read each word, normalize it to lowercase, and add it to the hashtable.

After loading the dictionary, the program should print out statistics on hashtable usage, and then repeatedly prompt the user for words, lowercase them, and check to see if they are in the dictionary.

2 Your implementation

The interface to your class matters. It should be easy to use, from the programmer's perspective and it needs to make sure to clean up after itself.

You should implement this using the *separate chaining* collision resolution strategy, using a `std::list<std::string>` object to store the contents of each bucket.

Your class should also provide a means for printing internal usage statistics, so we can gain some intuition about how well our hash functions distribute. This stats method should report the number of used and unused buckets, and the average, minimum (of the lists in use) and maximum bucket list lengths. Your driver code should calculate the wall clock time required to load the hash in each scenario.

3 Three hashing functions

You should implement the $37^i x^i$ monomial hashing function (using Horner's rule). You should also implement a straight ASCII addition method. Provide a third hashing function of your own design. These hashing functions should not know about the number of buckets. The class will have to take their values and modulo them to the appropriate range. Be sure to compensate for overflow values.

4 Report

In addition to running your program and demonstrating that it works properly, you will run several cases of the table-loading algorithm with different hashtable configurations. Report the stats for each of your hashing

functions with the following table sizes, as well as the next largest prime number for each size: 1000, 10000, 100000, 234936.

In paragraph form: interpret the results of your tests. Did using prime table sizes improve your hash function distribution? How did the straight ASCII sum do? Describe your custom hashing function. How did it do? Why are some functions better for different table sizes? How much did your hash function choice affect the time it took to run?

5 Ultrabonus (+50 pts)

Design a `HashTable` superclass, with your separate chaining table as a subclass. Provide a second subclass that implements one of the probing collision resolution strategies. Compare the performance of this scheme to the one that does separate chaining.

6 Turn In

Submit your report and code via blackboard by the due date.