CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean — Fall 2019
Webpage: `http://www.cs.clemson.edu/~bcdean/` — MW 2:30-3:45
Handout 3: Lab #2 — Daniel 313

1 Building a Spelling Checker Using a Hash Table

In this lab exercise, you will build a simple spelling checker using a hash table. You will find all the necessary files for this lab in the following directory; you should copy these files to your own directory space.

```
/group/course/cpsc212/f19/lab02
```

The files `stringset.h` and `stringset.cpp` define a C++ class that maintains a set of strings. This class uses the exact same interface as the integer set from lab #1, except now the keys are C++ strings instead of ints. The class supports the operations `insert`, `remove`, `find`, and `print` just as before.

2 First Goal: Complete the Stringset Class

Behind the scenes, the Stringset class is to be implemented using a hash table that uses chaining to resolve collisions. A Stringset maintains an array, `table`, of pointers to linked lists, so `table[h]` points to the beginning element of a linked list representing all strings hashing to value h.

If you look at the Stringset class, there are several functions already filled in but a few that are not. You will need to write code everywhere you find a "TBD" comment, in order to complete the implementation of the hash table. Note that a simple hash function is provided for you. Also note that the `print` function no longer prints elements in sorted order, since a hash table cannot easily maintain elements in sorted order. Accordingly, the linked lists in your hash table do not need to be kept in sorted order, as opposed to lab #1.

2.1 Testing

To test your Stringset class, compile it with `main.cpp`:

```
g++ -o main main.cpp stringset.cpp
```

You can then execute your code by running:

```
./main
```

As before, you can now enter commands like "insert word", "remove word", "find word", "print", and "quit". The testing interface should work just as in the previous lab.

2.2 Memory Issues

The Stringset keeps track of the size of its allocated table, as well as the number of elements stored in the table. As the number of elements increases, the performance of the `find` operation will degrade, since the average length of a linked list will grow large. To maintain good performance, it will therefore be necessary to expand the size of the hash table when it gets too full, defined here as when the number of elements stored in the hash table equals the size of the hash table.

The initial size of the hash table is currently set to 4 in the constructor. In the `insert` function, you will find an "if" statement:

```
if (num_elems == size) {
...
}
```

which is where you need to add the code for expanding the hash table. This code should allocate a new table of twice the size, re-insert all strings currently in the table into the new table, and then de-allocate the old table. To help with debugging, we recommend you leave this "if" statement untouched until you have all the other hash table functions working properly; note that the hash table will still work fine without this table expansion code, although the more elements you add, the slower it will get, since the average length of a linked list will grow larger and larger.

3 Second Goal: A Spelling Checker

In `main.cpp`, you will find two functions that main() can call. Currently, the function test() is called, and the function spellcheck() is commented out. The test() function provides you with the command-line testing code for using commands like "insert word" and "find word". Once you are confident that your hash table is working, comment out the call to test() and uncomment the call to spellcheck().

The spellcheck() function reads in the words in the file `words.txt` – approximately 50,000 of them – and inserts them all into your hash table (this will be a good test of how efficiently your insert function runs!). It then prompts you to type in one word at a time. For each word, it should suggest alternative words in the dictionary that differ by a change of exactly one character. For example:

```
Possible alternatives for word 'pointer':
painter
printer
pointed
```

If you type a word of length L, then there are exactly $25L$ alternative misspellings to check, since there are 25 ways to replace each character with alternatives. For each of these possible alternatives, you should use a call to `find` to check if it is present as a word in the dictionary.

If you have never used C++ strings before, they work much like strings in C with which you are probably more familiar: you can refer to the ith character in a string s by writing `s[i]`, and you can find the length of a string s by writing `s.length()`. Two strings can be compared for equality using the `==` operator.

4 Dining Preferences

An upcoming class demo will make use of data collected from everyone in the class in terms of their favorite restaurants in town. The file `dining-prefs.txt` contains a number of lines like

"Brian Dean" likes "All In Coffee Shop"

each describing a local restaurant that you like. Please modify this file so that it lists a small number (say, 3 to 5) of your favorite local restaurants, one per line, exactly in the format above. Then be sure to include this file in your submission.

5 Submission and Grading

Submit your `main.cpp`, `stringset.cpp`, and `dining-prefs.txt` files using handin.cs.clemson.edu just as with the previous lab. For this lab, you will receive 5 points for correct hash table code, 3 points for correct spelling checker code, and up to 2 extra points for good code: readability, comments, simplicity and elegance, etc. Zero points will be awarded for code that does not compile, so make sure your code compiles on the lab machines before submitting!

Final submissions are due by 11:59pm on the evening of Saturday, September 7. No late submissions will be accepted.

6 If Bored...

For those who finish early or are interested in more challenging extensions of the lab, consider the following optional ideas for fun:

- Would it be easy to modify your spell checker to account for other common misspellings, such as a single inserted character, a single deleted character, or two adjacent characters that are swapped?
- When looking for suggested spellings of a word of length L, each potential suggestion involves a hash lookup, which takes at least $\Theta(L)$ time. Based on the fact that we are using a polynomial hash function, can you think of a way to generate the hash of each new suggestion in only $O(1)$ time? As a hint, only one character is changing each time we generate a new suggestion.
- How quickly can you determine the word that generates the most suggested spelling corrections? Note that this word itself may not necessarily be a valid English word.

The instructor and TAs will be happy to discuss these questions with you if you like. But they are 100% optional; no need to turn anything in.