

# Csci 335 Assignment 3

---

*Due Thursday, April 6*

## **\*\*Programming: Hashing (100 points)**

The goal of this assignment is to test three hashing implementations. You will also use your best hashing implementation for a simple spell-checker.

Test each hashing implementation as follows:

- A) Read all words from a given text file words.txt and insert them into a hash table. After the table is created print out the total number of elements in the table (N), the size of table (T), the load factor (N/T), the total number of collisions (C), and the average number of collisions (C/N).
- B) Check whether each word in another given file query\_words.txt is in the hash table or not. For each word that is found print the word, the string "Found" and the number of probes used. For each word not found, print the word, the string "Not Found" and the number of probes used.

To implement the above, write a test program named **CreateAndTestHash**.

Your programs should run from the terminal as follows:

**CreateAndTestHash <words file name> <query words file name> <flag>**

<flag> should be "quadratic" for quadratic probing, "linear" for linear probing, and "double" for double hashing.

For example you can write on the terminal:

**CreateAndTestHash words.txt query\_words.txt quadratic**

### **Part1 (20 points)**

Modify the code for quadratic and linear probing and test CreateAndTestHash.

### **Part 2 (30 points)**

Write code to implement double hashing and test using CreateAndTestHash. This is going to be a variation of quadratic probing. The difference will lie in function FindPos() that has to now provide probes using a different strategy. As the second hash function you can try

the one described in the book:  $R - (x \bmod R)$  where  $R$  is a prime smaller than table size. Try various values for  $R$  and keep the one that provides the best performance (i.e. smaller number of collisions). In your Readme file state the exact  $R$  you are using.

Note that again you should provide templated testing functions so that testing code is not replicated.

### Part 3 (50 points)

Now you are ready to implement a spell checker by using your favorite hash table. Given a document your program should output all of the misspelled words. For each misspelled word you should also provide a list of candidate corrections from the dictionary that can be formed by applying one of the following rules to the misspelled word:

- a) Adding one character in each possible position
- b) Removing one character from the word
- c) Swapping adjacent characters in the word

Your program should run as follows:

**SpellCheck <document file> <dictionary file>**

You will be provided with a small document named `document1_short.txt` (and `document1.txt`) and a dictionary file with approximately 100k words named `wordsEn.txt`.

As an example, the following mistakes should be corrected by your spell checker:

complete -> complete (case a)

deciasion -> decision (case b)

lwa -> law (case c)

Note that there may be other words in the document that are not in the dictionary due to the limited size and scope of the dictionary. You do not have to correct those misspellings.

---

PS: You can add the keyword `mutable` in front of variables that you don't want to affect the external "costness" of a class method. For example

private:

```
mutable int number_of_collisions_;
```

```
// That means that the value of number_of_collisions_ can change even in const functions  
// of the class.
```