**CSCI 335** **Spring 2023**

# HW 3: Hashing

## 75 points for deliverables + 25 points for design

**Due date: April 3, 2023**

**Instructions:**

1. **Read and follow the contents of Programming Rules document on the blackboard (Course Information Section).**
2. **Submit only the files requested in the deliverables at the bottom of the description to Gradescope by the deadline.**
3. **Please note we will be using code comparison software across all submissions to ensure academic integrity requirements are strictly followed.**

**Learning Outcome: The goal of this assignment is hashing. It includes testing three hashing implementations. You will also use your best hashing implementation for a simple spell-checker. Acknowledge the sources you use in the README file**

## Q1: Hashing 75 points

We first provide a summary of what you will have to do for this question. The deliverables and output format will be provided at the end of the file. You will test each hashing implementation with the following:

A) The total number of elements in the table (*N*), the size of the table (*T*), the load factor (*N/T*), the total number of collisions (*C*), and the average number of collisions (*C/N*).

B) You will check whether each word in another given file, `query_words.txt`, is in the hash table, print the corresponding output depending on whether the word is found or not found, and how many probes it took to find the word if it exists. [Although you are provided with a file, we will use an unseen test file with another file name, which may contain any subset of words from words.txt.]

To implement the above, you will write a test program named **`create_and_test_hash.cc`**. Your programs should run from the terminal like so:

```
./create_and_test_hash <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:

```
./create_and_test_hash words.txt query_words.txt quadratic
```

For double hashing, the format will be slightly different, namely as follows:

**`./create_and_test_hash words.txt query_words.txt double <R VALUE>`**

The R-value should be xused in your implementation of the double hashing technique as discussed in class and described in the textbook: $hash_2$ `(x) = R - (x mod R).`

## Q1. Part 1 (15 points)

Modify the code provided for quadratic and linear probing and test `create_and_test_hash`. Note the code provided is from the textbook where it is also discussed in more detail.

Do **NOT** write any functionality inside the `main()` function within `create_and_test_hash.cc`. Write all functionality inside the `testWrapperFunction()` within that file. We will be using our own main, directly calling `testWrapperFunction()`. This wrapper function is passed all the command line arguments, as you would normally have in a main.

You will print the values mentioned in **part A** above, followed by queried words, whether they are found, and how many probes it took to determine so. ***The exact deliverables and output format are described at the end of the file.***

## Q1. Part 2 (20 points)

Write code to implement `double_hashing.h`, and test using `create_and_test_hash`. This will be a variation on quadratic probing. The difference will lie in the function `FindPos()`, which now has to provide probes using a different strategy. As the second hash function, use the one discussed in class and found in the textbook $hash_2$ `(x) = R - (x mod R)`. We will test your code with our own R-values. Further, please specify which R-values you used for testing your program inside your README.

Remember to **NOT** have any functionality inside the `main()` of `create_and_test_hash.cc`

You will print the current R-value, the values mentioned in **part A** above, followed by queried words, whether they are found, and how many probes it took to determine so. ***The exact deliverables and output format are described at the end of the file.***

## Q1. Part 3 (40 points)

Now you are ready to implement a spell checker by using a linear, quadratic or double hashing algorithm. Given a document, your program should output all of the correctly spelled words, labeled as such, and all of the misspelled words. For each misspelled word you should 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 any possible position
b) Removing one character from the word

c) Swapping adjacent characters in the word

Your program should run as follows:

```
./spell_check <document file> <dictionary file>
```

You will be provided with a short document named **`document1_short.txt`**, a longer document named **`document1.txt`**, and a dictionary file with approximately 100k words named **`wordsEN.txt`**.

As an example, your spell checker should correct the following mistakes.

comlete -> complete (case a)
deciasion -> decision (case b)
lwa -> law (case c)

**Correct any word that does not exist in the dictionary file provided (even if it is correct in the English language).**

**Some hints:**

1. **Note that the dictionary we provide is a subset of the actual English dictionary, as long as your spell check is logical you will get the grade. For instance, the letter "i" is not in the dictionary and the correction could be "in", "if" or even "hi". This is an acceptable output.**
2. **Also, if "Editor's" is corrected to "editors" that is ok. (case B, remove character)**
3. **We suggest all punctuation at the beginning and end be removed and for all words convert the letters to lowercase (for e.g. Hello! is replaced with hello, before the spell checking itself). You can assume the punctuation to be what is provided in the text documents provided.**

Do **NOT** write any functionality inside the `main()` function within `spell_check.cc`. Write all functionality inside the `testSpellingWrapper()` within that file. We will be using our own main, directly calling `testSpellingWrapper()`. This wrapper function is passed all the command line arguments as you would normally have in a main.

You will print the word, whether it is already spelled correctly (aka found), and all the possible spelling corrections if the word is not found in the dictionary. Be wary of punctuation and formatting in the document when parsing!

***The exact deliverables and output format are described below.***

## Q1 DELIVERABLES

You should submit these files:

1. **`README`** file (as usual, with extra information as requested)
2. **`create_and_test_hash.cc`** (modified and as specified in Part 1).
3. **`spell_check.cc`** (modified)
4. **`linear_probing.h`** (new)
5. **`quadratic_probing.h`** (modified)

6. **`double_hashing.h`** (new)

## Q1 FORMATTING

**For the linear and quadratic probing flags (Q1 Part 1), the format should be as follows**:

```
number_of_elements: <int>
size_of_table: <int>
load_factor: <float>
collisions: <int>
avg_collisions: <float>

<word1> Found <probes1>
<word2> Not_Found <probes2>
<word3> Found <probes3>
```

*Please include the single new line shown between the table properties section and the listing of the words. Please include the underscore between "not" and "found". The list of words shown here an example, the real output will depend on the words in query_words.txt*

*Important note: Your values for **collisions** and **avg_collisions** WILL VARY depending on the machine you use. Be aware of this when switching machines or asking for help. Your submissions will be graded on the same machine, so those values will be consistent as long as your implementation is correct for everything else. This will occur for ALL flags, linear, quadratic, and double.*

**Example of proper output (numbers are not representative of an actual table):**

```
number_of_elements: 500
size_of_table: 1350
load_factor: 0.37037037
collisions: 3
avg_collisions: 0.006

law Found 1
factz Not_Found 3
book Found 1
```

**For the double flag (Q1 Part 2), the format should be as follows:**

```
r_value: <int>
<SAME FORMAT AS LINEAR / QUADRATIC>
```

*The only difference between the output for double hashing is the addition of the line r_value: <int>. **Immediately below that line should be the same output as in linear and quadratic probing, described above**. Please include the underscore between "not" and "found".*

**Example of proper output (numbers are not representative of an actual table):**

```
r_value: 7
number_of_elements: 200
size_of_table: 750
load_factor: 0.266666
collisions: 3
avg_collisions: 0.015

law Found 1
factz Not_Found 3
book Found 1
```

**For spell checking (Q1 P3), the output should be as follows.**

```
<word1> is CORRECT
<word2> is CORRECT
<word3> is INCORRECT
** <word3> -> <alternate word> ** case <TYPE: A, B or C>
** <word3> -> <alternate word> ** case <TYPE: A, B or C>
<word4> is CORRECT
```

*Please make sure that your whitespaces and newlines are correct. Please include the ** and the -> in your output. The list of words here shown is an example, the real output will depend on the words in the document file. There may be more than one result per case type.*

**Example of proper output: (words shown are not representative of an actual input)**

```
decision is CORRECT
likely is CORRECT
interpretation is CORRECT
lwa is INCORRECT
** lwa -> wa ** case B
** lwa -> la ** case B
** lwa -> law ** case C
cases is CORRECT
```

**Finally, problems with your code? Here are reminders we have**

1. Double check your assignment matches the specifications as well.
2. Test your code on the Hunter lab machines remotely (yes even if it works perfectly on your machine) before uploading it to Gradescope
3. Ask questions & check the discussion board (Piazza).