CSCI 335 Spring 2021

Homework 4: Hashing

(85 points deliverables, 15 points design, total 100 points)

Due: April 9, 2021, 11pm
This is an individual assignment
Instructions:

1. **Read carefully and follow the contents of the Programming Rules document on Blackboard ('Course Information' Section).**
2. **This assignment has two components: program correctness (85 points) and design (15 points).**
3. **Submit only the files requested, and only the files requested, in the deliverables at the bottom of the description to Gradescope by the deadline.**

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

Description (85 points deliverables/correctness / 15 points design)

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 (Lambda = N/T), the average number of collisions (M = C/N), and the total number of collisions (C),
B) You will check whether each word in another given file, query_words.txt, is in the hash table, and print corresponding output depending on whether the word is found and 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 (the terminal prompt being %):

```
% ./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
```

You can use the provided Makefile in order to compile and test your code.

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

```
% ./create_and_test_hash words.txt query_words.txt double R
```

Or, alternatively use:

```
% ./create_and_test_hash words.txt query_words.txt double
```

Here R is the optional R value that should be used in your implementation of the double hashing technique discussed in class and described in the textbook: $hash_2(x) = R - (x \bmod R)$. A default R value should be specified in the code in case the user does not enter one on the command line.

I. Hashing with linear and quadratic probing (20 points)

Modify the code provided, for quadratic and linear probing and test create_and_test_hash.

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

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

II. Double hashing (25 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()`, that has to now provide probes using a different strategy. As the second hash function, use the one discussed in the class lectures and found in the textbook $hash_2(x) = R - (x \bmod R)$. 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.HASH file.

Remember to NOT have any functionality inside the main() function 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. ***Exact deliverables and output format are described at the end of the file.***

III. Spell checker (40 points)

Now you are ready to implement a spell checker by using a linear or 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 from the command line as follows:

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

You will be provided with a small document named **document1_short.txt**, **document_1.txt**, and a dictionary file with approximately 370k words named **wordsEnglish.txt**.
As an example, your spell checker should correct the following mistakes.

deciive -> decisive (Case A)
deciasion -> decision (Case B)
ocuntry -> country (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 lower case (for e.g. Hello! is replaced with hello, before the spell checking itself).**

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 function, directly calling `testSpellingWrapper()`. This wrapper function is passed all the command line arguments as you would normally have in a main function.

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!

Exact deliverables and output format are described below.

DELIVERABLES

You should submit these files:

1. **`README.HASH`** file (with your name, date, description of what you did, as usual, with extra information as requested)
2. **`create_and_test_hash.cc`** (modified and as specified in Part I).
3. **`spell_check.cc`** (modified)
4. **`linear_probing.h`** (new)
5. **`quadratic_probing.h`** (modified)
6. **`double_hashing.h`** (new)

FORMATTING

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

```
number_of_elements: <int>
size_of_table: <int>
load_factor: <float>
average_collisions: <float>
total_collisions: <int>
<new line>
<word1> Found <probes1>
<word2> Not_Found <probes2>
<word3> Found <probes3>
```

Please include the single new line (ASCII 10, listed as "<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

*Your values for **average_collisions** and **total_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: 670
size_of_table: 1560
load_factor: 0.429487
average_collisions: 0.005
total_collisions: 3

hill Found 1
skiny Not_Found 3
baked Found 1
```

For the double flag (Part II), 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
average_collisions: 0.0135
total_collisions: 3

hill Found 1
skiny Not_Found 3
baked Found 1
```

For spell checking (Part III), 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, capitalizations, 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)

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