Assignment 3: Word Count Wizard

Data Structures (CS-UH 1050) — Spring 2021

1 Code of Conduct

All assignments are graded, meaning we expect you to adhere to the academic integrity standards of NYU Abu Dhabi. To avoid any confusion regarding this, we will briefly state what is and isn't allowed when working on an assignment.

Any documents and program code that you submit must be fully written by yourself. You can discuss your work with fellow students, as long as these discussions are restricted to general solution techniques. Put differently, these discussions should not be about concrete code you are writing, nor about specific results you wish to submit. When discussing an assignment with others, this should never lead to you possessing the complete or partial solution of others, regardless of whether the solution is in paper or digital form, and independent of who made the solution, meaning you are also not allowed to possess solutions by someone from a different year or course, by someone from another university, or code from the Internet, etc. This also implies that there is never a valid reason to share your code with fellow students, and that there is no valid reason to publish your code online in any form. Every student is responsible for the work they submit. If there is any doubt during the grading about whether a student created the assignment themselves (e.g. if the solution matches that of others), we reserve the option to let the student explain why this is the case. In case doubts remain, or we decide to directly escalate the issue, the suspected violations will be reported to the academic administration according to the policies of NYU Abu Dhabi (see https://students.nyuad.nyu.edu/campus-life/community-standards/policies/academic-integrity/).

2 Introduction

In this assignment, you are expected to design and implement a Word Count Wizard! The system should be able to count the number of unique words in the entire text provided by the user, return the frequency of any word in that given text, among other operations. The wizard should take the input text and organize it in a hash table. Further instructions are described in the coming section.

You are supposed to create the system utilizing object-oriented programming (OOP) principles and appropriate data structures discussed in class. **You should implement all the data structures manually, <u>STL based containers are not allowed</u>.**

3 Implementation

Commands to implement:

Loading: When the Word Count Wizard starts, the user is greeted and asked to provide: 1) the full path of an input file and 2) an optional choice of a hash function (the user is presented with a list of four possible hash functions to choose from). If no choice is made (click enter), the default hash function will be used.

Analytics: Next, the interface will display the count of collisions, the count of unique words and the total count of words, followed by a list of possible actions to invoke by the user: 1) Find the frequency of a given word, 2) Find the word with the highest frequency (optional) , and 3) quit the system.

Instructions:

1. The system should be able to read and extract words (strings) from the input file and load them into a hash table.

2. Exploit the **hash table** data structure in a way that allows you to identify and store the unique words and their frequencies in the Wizard system. *You should determine the most suitable structure of an (key, value) entry within the hash table **on your own**.*

3. In your implementation of the hash table, each bucket has to be implemented as an **ordered map** (define and implement any necessary classes and their supporting methods). Use **separate chaining** to handle collisions.

4. Choose/design and implement **four different hash functions**, and determine which one should be set as the default hash function in the system.
   a. You have to evaluate the performance of each hash function in terms of the number of collisions when applied on the same data (use the supplementary files). The hash function causing the least number of collisions, on average across all the files, should be set as the default hash function to use by your system.
   b. You have to document the evaluation results of each hash function in a pdf report.
   c. A user should be able to choose one of the other implemented hash functions to be used by the system instead of the default one if they choose to do so.
      i. Implement a **switch_hashFun** method to take care of this

5. Implement the **count_collisions** method: returns the number of collisions caused by a hash function.

6. Implement the **count_unique_words** method: returns the number of unique words in the entire text.

7. Implement the **count_words** method: returns the total number of words in the text.

8. Implement the **find_freq(word)** method: searches for a word and returns its frequency, if it is found. Otherwise, it will return zero.
    a. Implement an “efficient” **search** method (O(log n)) for ordered maps to search within a bucket.

9. (Bonus 1 point 🙆) Implement the **find_max()** method: returns the word with the highest frequency
    a. Note: this will require a new separate data structure, a **Heap**, to represent the set of entries.

Hints for text tokenization:

- Split each sentence by whitespace in order to extract words.
- Ignore punctuation marks (“.”, “,”, “\” , “!”, “?”, “;”) attached to the end of a word.
- Ignore cases (you may use the function tolower() for that purpose)
    - For example, "Token?" should be counted as an instance of "token".

Dataset:

- We included a large collection of stories (101) in a TXT format to test your work, and evaluate your hash functions.

3 Grading

Description	**Score (/20)**
Initializing the Word Count Wizard properly: - Proper (informative) display of all messages expected from the system - Loading and reading the input file correctly - The main functions are properly displayed and executed - Proper quitting of the system	4
Clear and complete report of the results exploring the performance of the different hash functions in terms of the average number of collisions when applied to the same set of text files (report the results per function).	4
Defining and implementing all classes and methods correctly, so that all the functionalities are working properly	8
Proper error handling of missing and invalid input, etc.	2
Documentation (code comments)	2

Note: Extra points (🙋) are used to pad your score up to the maximum score, but the total cannot exceed 20 points.

You should solve and work **individually** on this assignment. The deadline of this assignment is **the NYUAD last day of classes (May 3rd)** on NYU Brightspace.

No Late submissions will be accepted. All assignments are due at 11:59pm on the due date.

You should directly submit your pdf document, C++ source files (*.cpp, *.hpp) and Makefile (if any) on NYU Brightspace. Submissions via email are unacceptable.

Note that your system should be implemented in C++ and **must be runnable on the Linux, Unix or macOS operating system**.