Text Analyzer Project Description

Natalie Kalinowski, Colleen Dunlap, Paula Van Camp, Matthew Petter

Abstract - **Our program is a Text Analyzer which will serve the purpose of searching documents in order to find repetitive language. The output of the program will depend on the user's selection of either returning keywords to retain from the documents or repetitive language for revision and improvement. The solution of this problem will take advantage of hash tables to store key value pairs. Frequency and density will be analyzed by looking at where the most frequent words appear within our sorted data structure. The Text Analyzer is a perfect tool for technical and creative writers, which provides an objective critique of repetitive language. It also functions as a learning strategy by giving a clear and simple set of keywords in a given set of texts.**

I. Introduction

The Text Analyzer is a program that searches documents for repetitive language and, based on the user's selection, will return either keywords to retain from the documents (learn) or suggested revisions to repetitive language (revise). The program will sort all the words based on frequency and give the user a clean list of words. Because words like 'and', 'the', 'or', and other connective/necessary language appears so frequently, they will be ignored - the word conditions are described in more detail within section II. This program gives the frequent terms in a list. If the user selected "learn" initially, the program will give definitions along with each word in the list to read and retain. If the user selected "revise" then the list will be paired with synonyms to modify repetitive language.

II. Proposed Approach

The Text analyzer will be constructed using Python. The solution of this problem will take advantage of hash tables to store key value pairs, where the "key" is the word and the "value" is the number of occurrences of that word within the text. We will create our own Hash Table, and develop a Hash Function to ensure there are as few collisions as possible[2]. Each word - defined by a group of 1 or more characters separated by a space - will be considered individually and then processed accordingly. The criteria for retaining a word in our data structure will be that it is 3 or more characters, or 3 or less characters and contains lowercase. Anything less than 3 characters is most likely a connecting word that is frequent but necessary. The other possibility is that the word is an important abbreviation, which is why case of letters is considered. In general, frequency and density will be considered and tracked throughout the sorting process. In addition to our main word-occurence storage in the form of a Hash Table, we will also have a Doubly Linked List of the 20-40 most common words (proportional to the size of the file)[3]. A variable will track the occurrence of the least common word in the List. When a word's occurence is incremented in the Hash Table, it's occurrence will be compared to the value of the aforementioned variable. If it is greater than that variable's value, it will be inserted into the Doubly Linked List from the "back," in a process that reflects a sub-problem of the Insertion Sort algorithm with a Linked List implementation (an "outside" element is being taken and inserted into the proper place). Once the strings have all been sorted, each word will be output with their corresponding synonyms or definition using the py-thesaurus API [4]. The user input will be checked and the corresponding method will be called. If the input was "learn" each item in the list will return along with the definition. If the input was "revise" then the API will be used to source synonyms and output for the user. Output will be in the form of text in the command line.

III. Project Timeline and Milestones

Work on program set-up and research will kick off the project following the submission of this proposal. The progress of the project will be tracked weekly leading up to the completion date on May 1st. The program functionality and correctness is most important and will be considered complete by April 11. Ideally, the remaining time after this date will be used to implement a user interface, however if obstacles arise, this time will be used to complete any outstanding tasks.

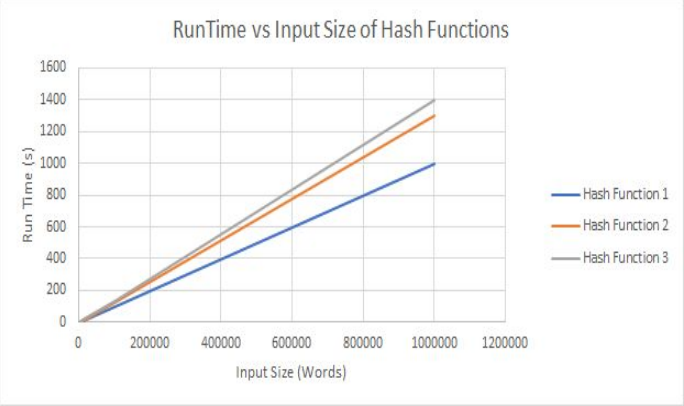
Table I
PROJECT TIMELINE AND MILESTONES

Task	Expected Conclusion	Milestone
Decide Platform and design program architecture	2/18	Begin Project
Research and implement text importing method	2/25	Data Import
Implement hash table functionality	3/4	Data Sort
Output most repetitive words	3/18	Output
Integrate resource for finding alternate words and defs	3/25	Word Replacement
Implement, test, and evaluate project prototype	3/28	Project Prototype
Write tests	4/4	Test
Evaluate Correctness	4/11	Performance
Implement User Interface (If time allows)	4/18	GUI
Compile results and prepare project poster	5/1	Project Poster

IV. Evaluation Plan

The correctness and evaluation of the program will be completed by writing functional tests of varying size and complexity. For small input files, the output can be validated through counting and human analysis. For larger inputs, we will have separate test cases that output strings in a random hash value along with their occurrences and use a pdf tool to count/confirm their occurrences in a different environment. In general, the run time of storing all the text data in a Hash table is ***O***(n). Since we are storing the maximum occurrences into the doubly linked list using insertion sort, the time complexity is dependant on the occurrence (minimum of 1 instruction or a max of 40), but evaluates to constant time. Once the sort is complete, each piece of data will be processed and output once so the final output method will take ***O***(n). Overall, the program will be no more efficient than ***O***(n). We can improve the run time of our algorithm by optimizing our Hash Function to avoid collisions. We will compare multiple hash functions to determine the runtime of the program with increasing input size.

Example Efficiency Analysis Plot



V. References

[1]Project Repository: https://github.com/colleenDunlap/Writing-Buddy

[2] Cs.cmu.edu, *Concept of Hashing,* 2019.. Available at: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Hashing/hashing.html [Accessed 11 Feb. 2019]

[3]Hash Functions and Data Structures: Dale, Nell, et al. *Object-Oriented Data Structures Using Java*. Jones & Bartlett Learning, 2018.

[4]Redpills, 'py-thesaurus 1.0.5', 2018. [Online]. Available: https://pypi.org/project/py-thesaurus/. [Accessed: 11-FEB-2019].