Kingdom of Saudi Arabia
King Saud University
College of Computer & Information Sciences
Department of Computer Science

Word Analysis Tool
For Unstructured Dataset

Submitted By :

Amal Almuarik - 442200454
Lama Alzakri - ~~441204526~~

Section Number: 55027
Course Number: CSC212

Under the supervision of :
Dr. Lubna Alhinti.

Friday, 7 October 2022

ACKNOWLEDGMENTS

We would like to express our great gratitude to Dr. Lubna Alhinti for her valuable suggestions and aid throughout this report's writing. Her willingness to give her time so generously has been very much appreciated.

Contents

Introduction

1.1 Problem Statement

This work aims to develop a text analysis tool. Capable of analyzing unstructured text data to extract valuable information. In essence, the primary purpose is to design a suitable ADT (an abstract data type) for the problem to provide the user with a conceptual picture (the user's view of what an object looks like, how the structure is organized … etc.).

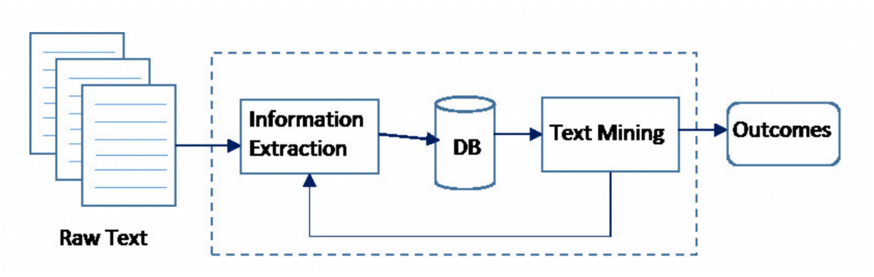


Fig.1. Overview of a Text Analysis Framework

1.2 Goals And Objective

- Organizing data in an appropriate manner to formulate efficient algorithms.
- Composing an abstract mathematical model for particular data structures.
- Calculating the complexity of the algorithm while manipulating the data with certain operations. In terms of the size of its inputs.
- Extracting valuable information from raw data with consistency.

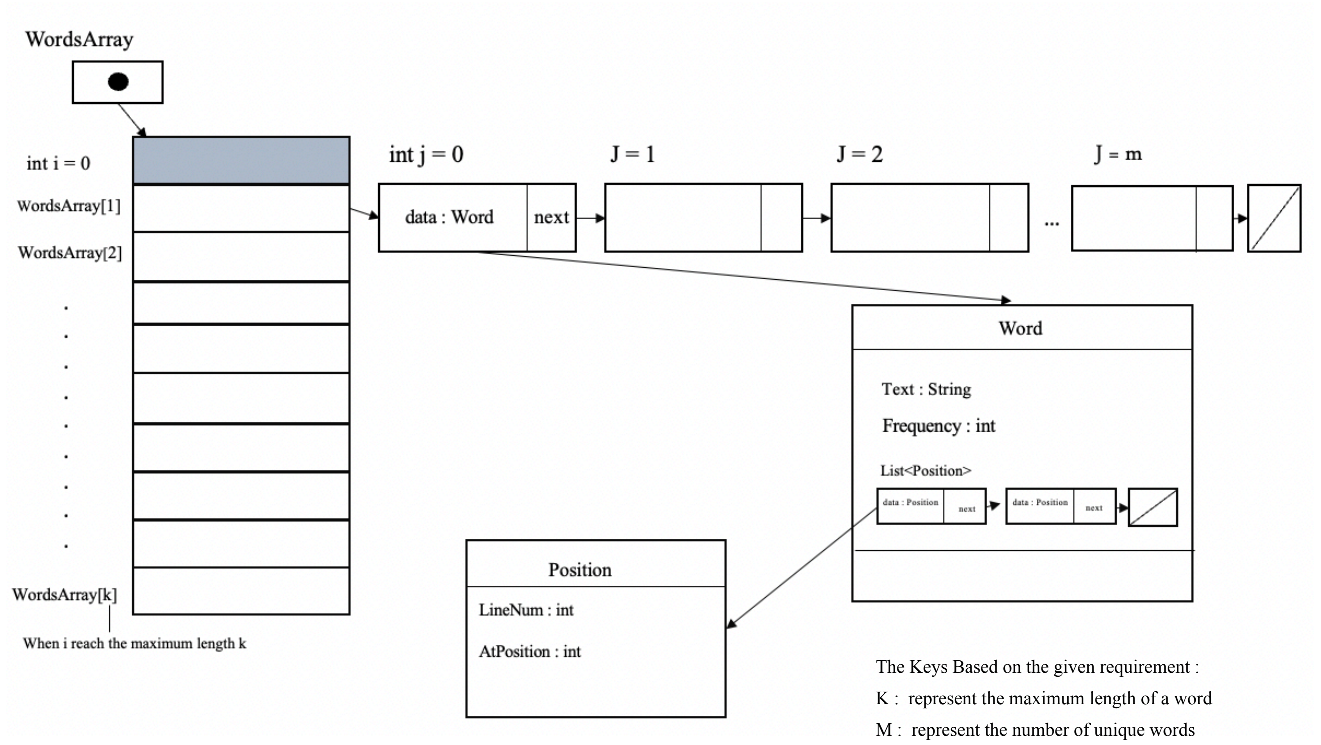
1.3 The DataSet

The data set we are going to use in this project will be provided by the user as a text file.

Background

2.1 Graphical Representation

The graphical representation is a combination of data structures, conceptually it is represented as



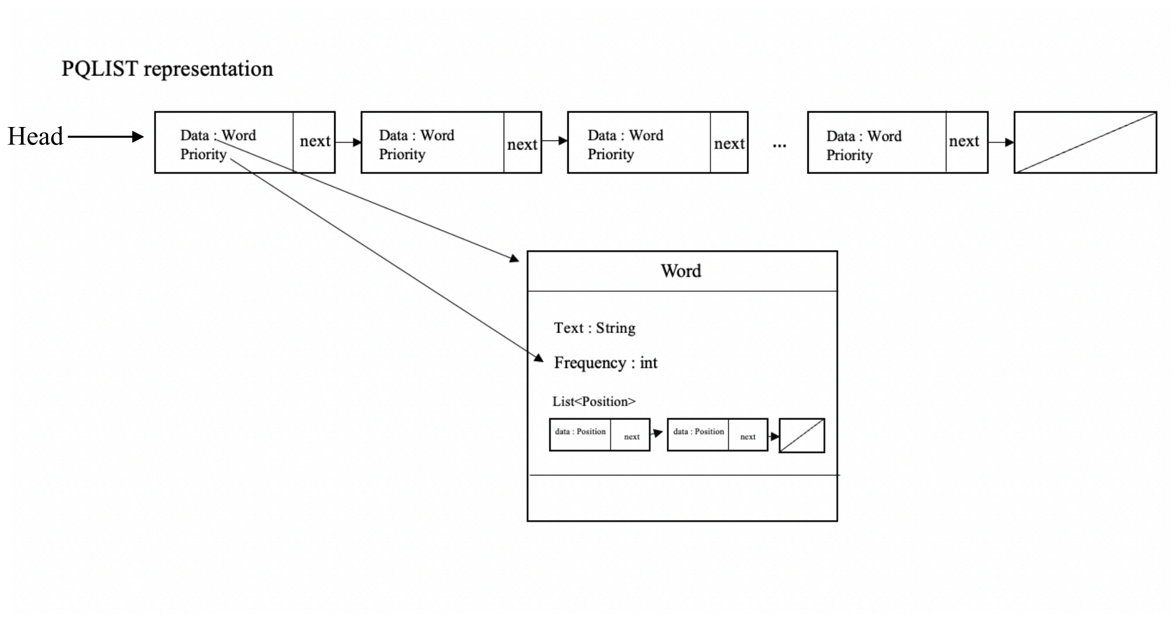
The Keys Based on the given requirement :

K : represent the maximum length of a word

M : represent the number of unique words

The Assumption : The first index of the array won't be used since there is no word of length 0 .

Fig.2. Schematic representation of ADT



2.2 An Overview Of the Suggested Data Structure

An Array of linked lists is an exciting design combining static and dynamic structures, which are an array, and a linked list, to form a functional data structure.

This data structure is used in many applications where we know the category of certain items, and we want to insert an unknown number of items based on their variety. This design, in particular, is suitable for the problem statement, especially If the primary purpose was to enhance the time complexity while not considering the space complexity.

Briefly, The process is as follows we can define an array and categorize each word based on its length and let the indices of the array represent the length of specific words, and we can assume later on that the size of the array is a constant number which holds the maximum length a word can reach and define it as k.

Each chunk inside the array will hold a pointer that references a list of pointers, where each pointer (Node) has a data attribute and a next pointer. The node's data is of a word type that stores inside it the word frequency and the list of positions for that single word, where the position of a word consists of both the line number and the exact location a word occurs in that particular line.

The reason for choosing this design and not others is that an appropriate measurement for the time complexity bounds the algorithms. Especially in the insertion process, we try to insert a list at a specific index by accessing the memory randomly with a constant time. And the time taken to search for a word in the worst-case scenario is a big order of n, which is not too massive compared to other designs.

Lastly, what is worth mentioning is that after all the insertions, we would use a particular type of queue known as a priority queue to sort the word by their frequency which is the word priority in descending order(from highest to lowest) to be able to display them in sorted.

2.3 System Specifications

Abstract Data Type "WordAnalysisADT"

On an abstract level, The word analysis ADT can be constructed with the following operations as follow :

- Total_Number_Of_Words();

  Input: None.

  Constraint: None.

  Output: An integer value represents a text file's total number of words.
  And if there is no word present, it returns 0.

  Time Complexity: O(1).

- Total_Number_Of_Unique_Words();

  Input: None.

  Constraint: None.

  Output: returns an integer value that counts only the unique words in a file.

  Time Complexity: O(1).

- Total_Occurrence_Of_A_Word(String word);

  Input: A String Value

  Constraint: None.

  Output: it returns the total occurrence of a particular word as an integer value and returns 0 only if the list was empty.

  Time Complexity:

  Case 1: if the length of the words was evenly distributed.

  O(n/k**)** where n is the total number of words and k is the number of unique words (**$k \neq 0$** *this condition is never met since no length is equal to 0*)

  Note: *we are generalizing the case by involving n and k as if the values are unknown but it's obvious if n, k is known it will be O(1) .*

  Case 2: if the length of words were not evenly distributed **O(**m**).**

- Words_With_Given_Length (int length)**;**

  Input: An integer value.

  Constraint: Non-empty list. And the received value is valid.

  Output: An integer value that counts the total number of words with a length equal to the received length.

  Time Complexity:

  Case 1: if the words were evenly distributed (E.g. Number of words with a length of 3 is equal to the number of words with a length of 5 … etc). O(1)

  Case 2: if the words were not evenly distributed. O(1)

- Display_Words_ Occurrences ()**;**

  Input: None.

  Constraint: Non-empty list.

  Output: It displays The unique word associated with its occurrence in descending order (From the most frequent to the least). Where each word is retrieved from a PQ.

  Time Complexity: O(m).

- Display_WordPosition_Given_Line(List<String> lines, int line_Num , String word);

  Input: it receives a linked list containing one line of words, a line number, and the word to be looked for.

  Constraint: Non-empty list.

  Output: It displays The word occurrences associated with their positions after processing each line individually and then joining all the lines to look for the bigger list of all lines.

  The above process might require a helping function to process all lines together but it won't affect the time complexity since it will be the sum of lines where each line contains a bunch of words. So the upper bound is bound by a big order of n in the worst-case scenario because we have to traverse through all words. But not more than O(n).

  Time Complexity: O(n).

- IsAdjacent(String word1 , String word2);

  Input: it takes two strings words and check whether they are adjacent or not.

  Constraint: Non-empty list.

  Output: returns a boolean value that indicates the truth value of the predicate.

  Time Complexity: O(*n*).

- LoadFile(String fileName);

  Input: it takes the filename that is read from the user

  Constraints :

  - The size of the file can not exceed the buffer default size.
  - The given path indicates a file but not a directory.
  - the existence of a file
  - The given File name must be followed by the extension of the file (E.g. ".txt")

  Process: Reading from the text file and ignoring all (white spaces, punctuations, … etc)

  In addition, Storing unique words as an object of type word and passing the value as node data to the linked list.

  Output: None.

An Additional specification just to clarify the process of displaying the word it will be after they are sorted

- PQSort();

  Input: None.

  Constraints: The received list is a non-empty list.

  Process: Sort all the list of words according to their occurrences in the file by inserting the elements in PQ List using the word frequency as its priority value and then returning the result back to the list of words.

  Output: None.