GTU Department of Computer Engineering
CSE 222/505 - Spring 2023
Homework #6 Report

REŞİT AYDIN
200104004019

1. Problem Definition

In the context of natural language processing, it is often necessary to process large amounts of text data and extract meaningful information from it. One important task is to identify the most frequently occurring words in the text. This can be achieved by counting the frequency of each word in the text and then sorting them in descending order of frequency.

To solve this problem, a Java program is required that takes in a string of text, preprocesses it by removing any special characters or punctuation, and then counts the frequency of each word in the text. The program should then output a sorted list of words which is a type of map along with their corresponding frequencies.

2. Problem Solution Approach

The problem of analyzing the frequency of words in a text is addressed through a solution that involves four classes: MainClass, info, myMap, and mergeSort. MainClass serves as the starting point of the program, taking the input text, cleaning it, and passing it on to the other classes for further analysis. The info class represents a collection of words and their corresponding frequency counts, offering methods for adding words to the list, retrieving the current count, and displaying the list of words.

The myMap class acts as a container for a HashMap that connects each word in the text to an info object holding its frequency count. This structure facilitates quick access to the count of each word during the sorting process. Finally, the mergeSort class implements the merge sort algorithm to organize the list of words in descending order of frequency. It does this by sorting the list based on their associated counts in the myMap object. The MergeSort() method is a public method that acts as an entry point for the merge sort algorithm. The private MergeSort(Character[] input) method is the core of the merge sort algorithm and it implements the divide-and-conquer strategy. It takes an array of characters as its input argument and recursively divides the array into two

halves until the length of the array's length is less than 2. Once the array is divided into two halves, the merge() method is called to merge the two sorted sub-arrays.

The merge() method takes three arguments: the original input array, the left half of the array, and the right half of the array. It then merges the left and right halves in ascending order based on the count of each character in the originalMap. Finally, the sorted aux array is populated with the sorted characters from the original array.

Once the merge sort algorithm is complete, the sorted aux array is used to create a new myMap object with sorted entries, and this sorted map is returned by the getSortedMap() method.

In essence, the solution consists of preparing the input text by removing punctuation and special characters, tallying the frequency of each word, and arranging the resulting list of words by their frequency. This approach permits the efficient processing of significant volumes of text data and can be enhanced with additional features or capabilities.

3. Running Commands and Outputs

3.1 Running Commands

```
resitaga@resitaga:~/Desktop/HW6_RA$ cd hw6
resitaga@resitaga:~/Desktop/HW6_RA/hw6$ javac *.java
resitaga@resitaga:~/Desktop/HW6_RA/hw6$ cd ..
resitaga@resitaga:~/Desktop/HW6_RA$ java hw6.MainClass
```

3.2 Outputs

```
Original String:        Buzzing bees buzz
Preprocessed String:    buzzing bees buzz


The original (unsorted) map:
Letter: b - Count: 3 - Words : [buzzing, bees, buzz]
Letter: u - Count: 2 - Words : [buzzing, buzz]
Letter: z - Count: 4 - Words : [buzzing, buzzing, buzz, buzz]
Letter: i - Count: 1 - Words : [buzzing]
Letter: n - Count: 1 - Words : [buzzing]
Letter: g - Count: 1 - Words : [buzzing]
Letter: e - Count: 2 - Words : [bees, bees]
Letter: s - Count: 1 - Words : [bees]



The sorted map:
Letter: i - Count: 1 - Words : [buzzing]
Letter: n - Count: 1 - Words : [buzzing]
Letter: g - Count: 1 - Words : [buzzing]
Letter: s - Count: 1 - Words : [bees]
Letter: u - Count: 2 - Words : [buzzing, buzz]
Letter: e - Count: 2 - Words : [bees, bees]
Letter: b - Count: 3 - Words : [buzzing, bees, buzz]
Letter: z - Count: 4 - Words : [buzzing, buzzing, buzz, buzz]
```

```
The original (unsorted) map:
Letter: h - Count: 7 - Words : [hush, hush, hush, hush, whispered, the, rushing]
Letter: u - Count: 3 - Words : [hush, hush, rushing]
Letter: s - Count: 4 - Words : [hush, hush, whispered, rushing]
Letter: w - Count: 2 - Words : [whispered, wind]
Letter: i - Count: 3 - Words : [whispered, rushing, wind]
Letter: p - Count: 1 - Words : [whispered]
Letter: e - Count: 3 - Words : [whispered, whispered, the]
Letter: r - Count: 2 - Words : [whispered, rushing]
Letter: d - Count: 2 - Words : [whispered, wind]
Letter: t - Count: 1 - Words : [the]
Letter: n - Count: 2 - Words : [rushing, wind]
Letter: g - Count: 1 - Words : [rushing]



The sorted map:
Letter: p - Count: 1 - Words : [whispered]
Letter: t - Count: 1 - Words : [the]
Letter: g - Count: 1 - Words : [rushing]
Letter: w - Count: 2 - Words : [whispered, wind]
Letter: r - Count: 2 - Words : [whispered, rushing]
Letter: d - Count: 2 - Words : [whispered, wind]
Letter: n - Count: 2 - Words : [rushing, wind]
Letter: u - Count: 3 - Words : [hush, hush, rushing]
Letter: i - Count: 3 - Words : [whispered, rushing, wind]
Letter: e - Count: 3 - Words : [whispered, whispered, the]
Letter: s - Count: 4 - Words : [hush, hush, whispered, rushing]
Letter: h - Count: 7 - Words : [hush, hush, hush, hush, whispered, the, rushing]
```