Gebze Technical University

Department Of Computer Engineering

CSE 222/505 - Spring 2023

Data Structures and Algorithms

Homework #6

Zeynep Çiğdem Parlatan

1901042705

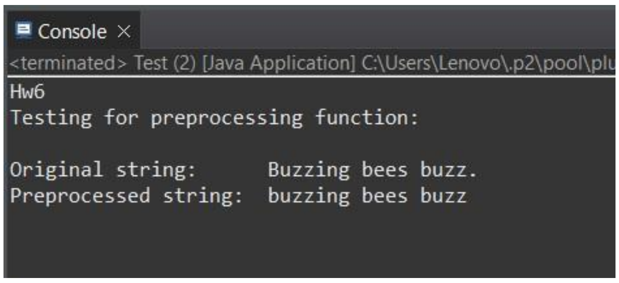
1.Problem Solutions

Within this assignment, we are expected to transform a given string into a suitable format (preprocessing string) and obtain a map according to a specific map structure. Then, we are expected to sort this map using the mergesort algorithm.

a. Step 1- Preprocessing;

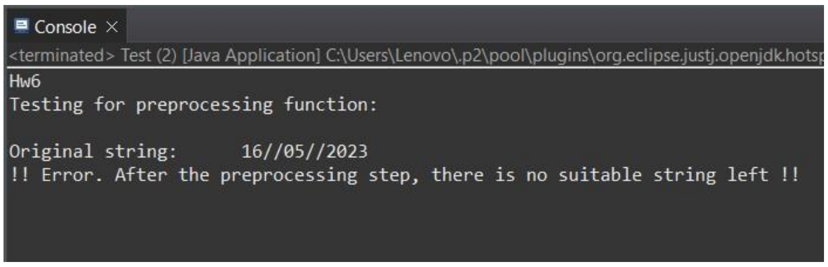
In this step, I looped through the given string for its entire length. During the loop, I examined each character in the string. If the character was a lowercase letter or a space, I directly assigned it to a temporary variable. If the character was an uppercase letter, I converted it to lowercase and assigned it to the temporary variable. I skipped all other characters. After these operations, if a valid string remained, I assigned it to the 'str' variable of my 'myMap' class and returned 0 from the function. If the length of the string became zero or the string only contained whitespace after the operations, I returned 1 from the function and printed an error message to the console.

- ❖ Original String: Buzzing bees buzz. => Converting ‘B’ to ‘b’ and remove ‘.’



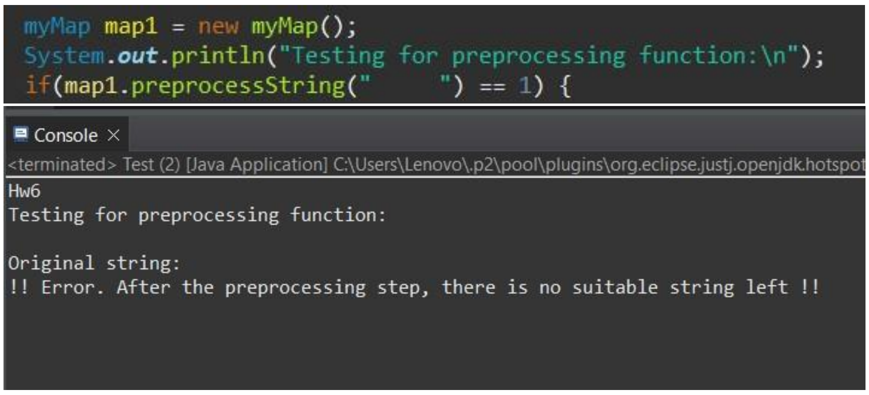
```
Console ✕
<terminated> Test (2) [Java Application] C:\Users\Lenovo\.p2\pool\plu
Hw6
Testing for preprocessing function:

Original string:      Buzzing bees buzz.
Preprocessed string:  buzzing bees buzz
```

- ❖ Original String: 16//05//2023 => No lowercase or space characters here. All will be removed. So the string length will be zero



```
Console ✕
<terminated> Test (2) [Java Application] C:\Users\Lenovo\.p2\pool\plugins\org.eclipse.justj.openjdk.hotsp
Hw6
Testing for preprocessing function:

Original string:      16//05//2023
!! Error. After the preprocessing step, there is no suitable string left !!
```

- ❖ Original String: (just space character) => There are only whitespace characters. I also give an error for this.



```java
myMap map1 = new myMap();
System.out.println("Testing for preprocessing function:\n");
if(map1.preprocessString("      ") == 1) {
```

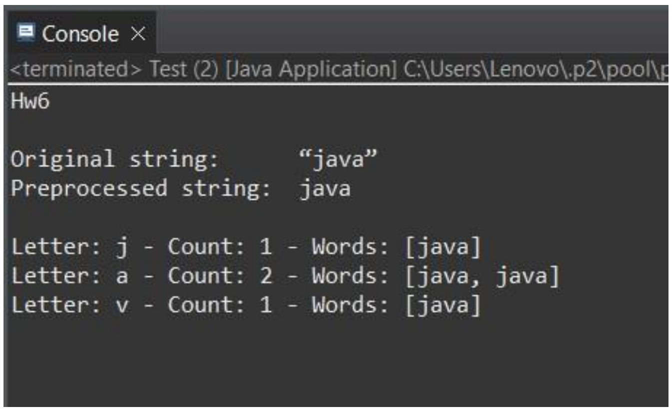
```
Console ✕
<terminated> Test (2) [Java Application] C:\Users\Lenovo\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot
Hw6
Testing for preprocessing function:

Original string:
!! Error. After the preprocessing step, there is no suitable string left !!
```

b. Step 2-Build map;

In this section, I am using the preprocessed string to create a map based on this string. We will use the LinkedHashMap<String, info> structure to create this map. So, our key is a string, and our value holds an object from the 'info' class. In the 'info' class, we have a 'count' variable that keeps track of the number of occurrences of the key and a 'words' variable that holds the words where the key appears. For the 'words' variable, I used the ArrayList structure. To create the map, first, I looped through each character in the string based on the length of the string. Here, each character is a key. For each character, I checked if it has already been added to the map. If it has not been added, I created an 'info' object, saved the count and words for this key, and then added this key and value to the map. If it has already been added, I just updated it in the map. In other words, I only changed the 'count' and 'words' variables for that key. This way, I obtained my map.

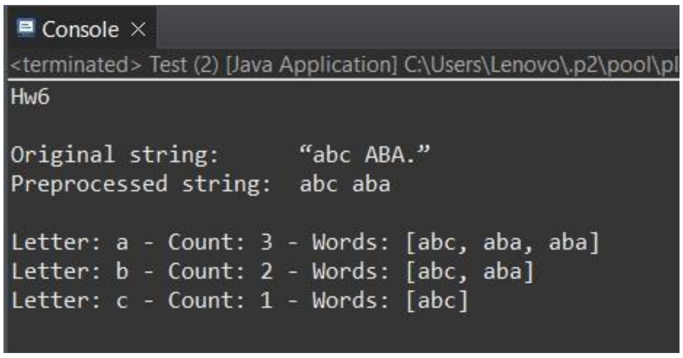
- String getWord(int index) => In this section, I also used a helper function. I used this helper function to extract the word where the key is found from the string. I pass the index where the key is found. In the function, I return the word where this index is located. For this, I look at the length of the string and the index of any space characters to get the word where the index is located.

- Test1:



```
Console ×
<terminated> Test (2) [Java Application] C:\Users\Lenovo\.p2\pool\p
Hw6

Original string:      “java”
Preprocessed string:  java

Letter: j - Count: 1 - Words: [java]
Letter: a - Count: 2 - Words: [java, java]
Letter: v - Count: 1 - Words: [java]
```

- Test2:



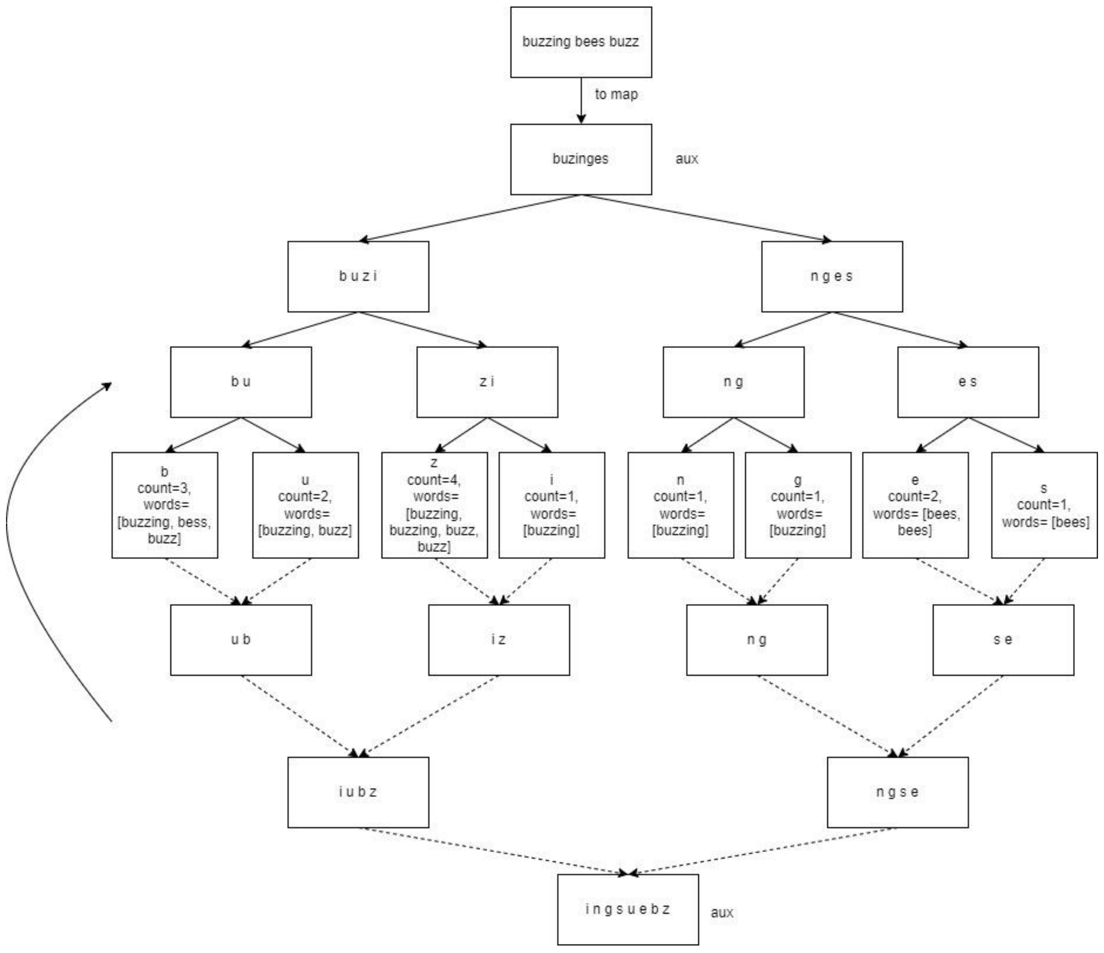
```
Console ×
<terminated> Test (2) [Java Application] C:\Users\Lenovo\.p2\pool\pl
Hw6

Original string:      “abc ABA.”
Preprocessed string:  abc aba

Letter: a - Count: 3 - Words: [abc, aba, aba]
Letter: b - Count: 2 - Words: [abc, aba]
Letter: c - Count: 1 - Words: [abc]
```

c. Step 3-MergeSort;

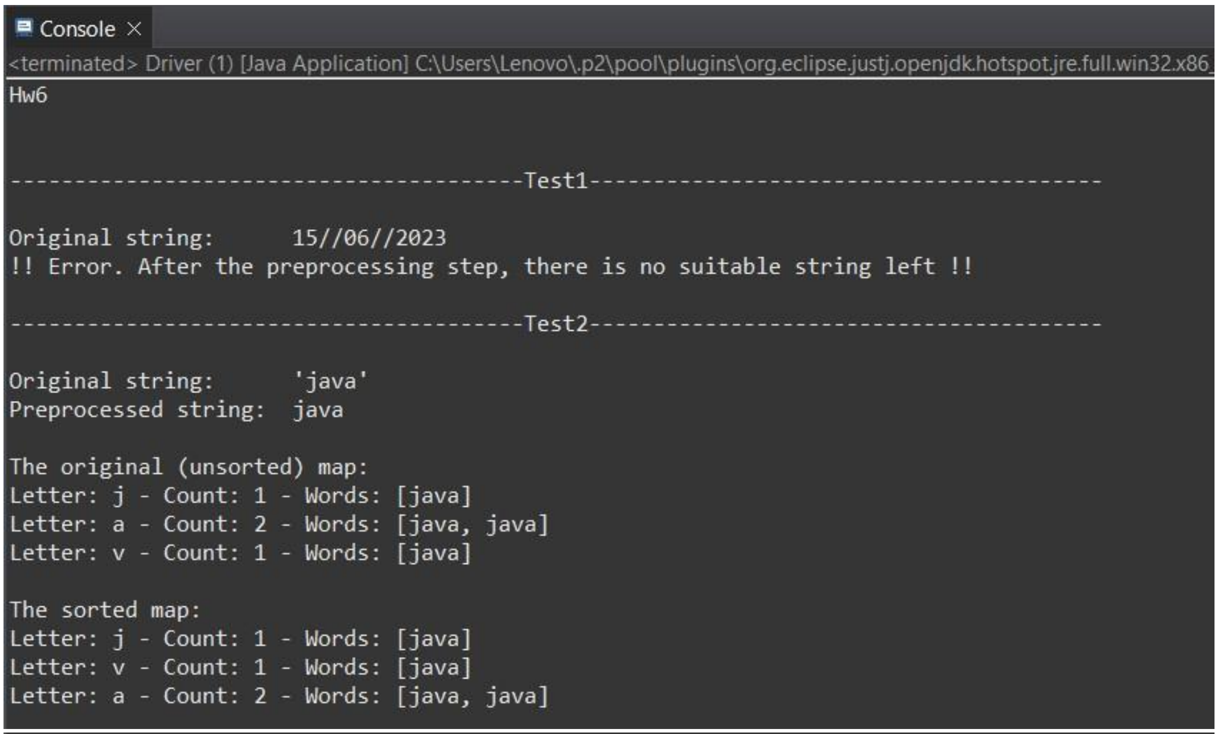
In this section, after bringing the given string into the appropriate form, we are asked to sort the created map based on the count values of the keys. We will use the merge sort algorithm for the sorting process. In the merge sort algorithm, we employ the divide and conquer approach. That is, we first divide the given unordered list into smaller parts, perform the sorting operation on each part, and then combine these sorted parts to obtain the entire list.

When I create an object from the mergeSort class, I use the constructor to save the keys into the String[] aux array based on the originalMap. For this purpose, I used a helper function called fillAux() to store the keys of the originalMap in the array. I pass this array to the mergeSort function. In the mergeSort function, our goal is to divide until the smallest parts remain, so I recursively call the function until only one element is left by dividing it in half each time. Then, in the merge function, I pass the aux array, the divided left and right arrays. In this part, we will sort based on the count values. I keep two integer arrays, leftKeyCount and rightKeyCount, for the count values. I store the count values corresponding to the keys in the left and right arrays in these arrays. Then, I sort and save them in the aux array based on these count values. After all the sorting and merging operations are done, the aux array contains the keys sorted according to their count values. Accordingly, I save these keys and their values in the sorted map. I then print the sorted map to the screen using the printMap() function.

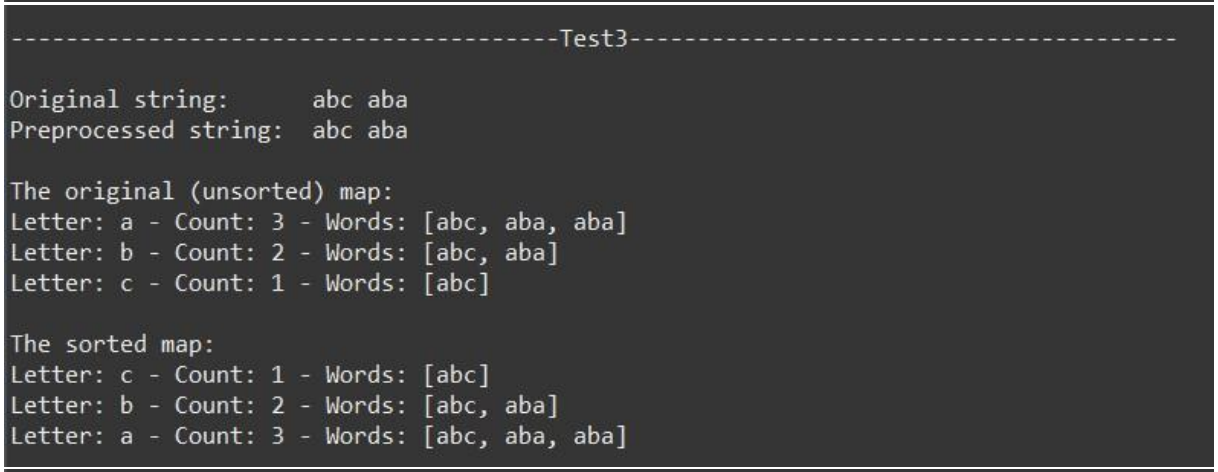
We can provide an example to illustrate the working logic as follows. For instance, if the string is 'Buzzing bees buzz', I apply step 1 and step 2 to it. I assign this map to the originalMap value of the object created from the mergeSort class. The divide operations are performed, and the parts are sorted internally based on the count value. Eventually, we obtain the sorted version of the string.


buzzing bees buzz
to map
buzinges
aux
b u z i
n g e s
b u
z i
n g
e s
b count=3, words= [buzzing, bess, buzz]
u count=2, words= [buzzing, buzz]
z count=4, words= [buzzing, buzzing, buzz, buzz]
i count=1, words= [buzzing]
n count=1, words= [buzzing]
g count=1, words= [buzzing]
e count=2, words= [bees, bees]
s count=1, words= [bees]
u b
i z
n g
s e
i u b z
n g s e
i n g s u e b z
aux

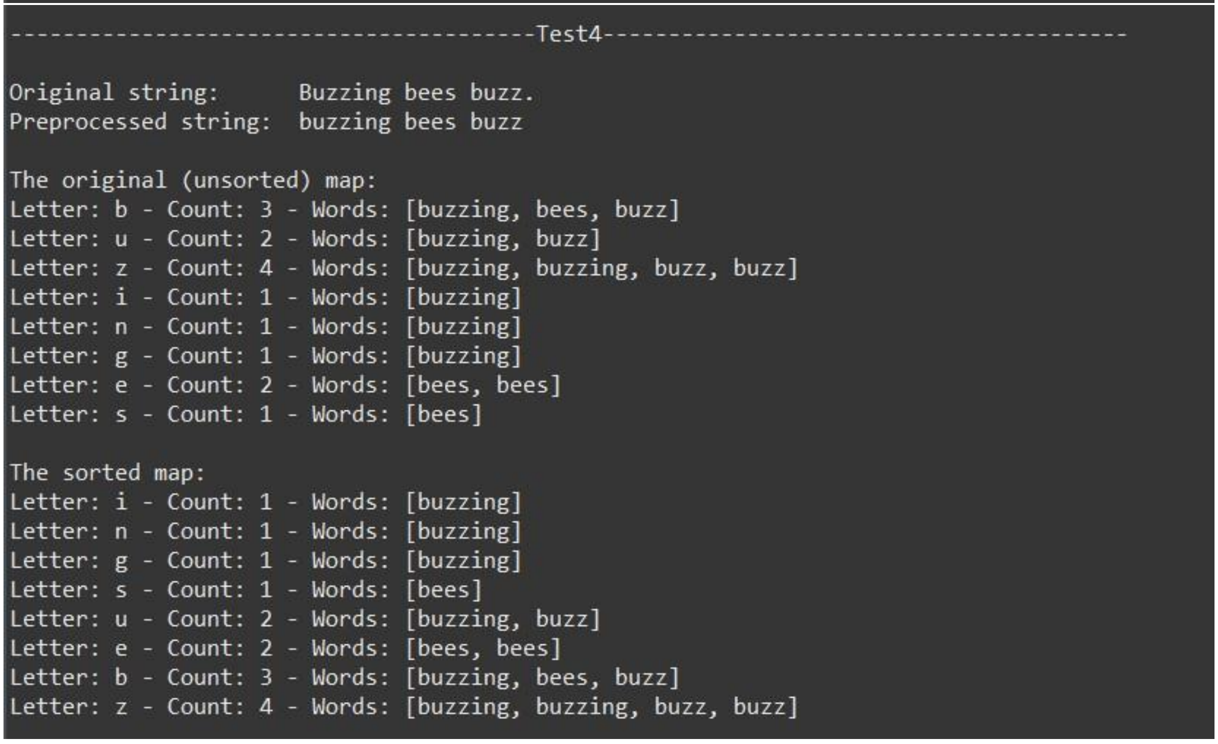
2.Running Command and Results



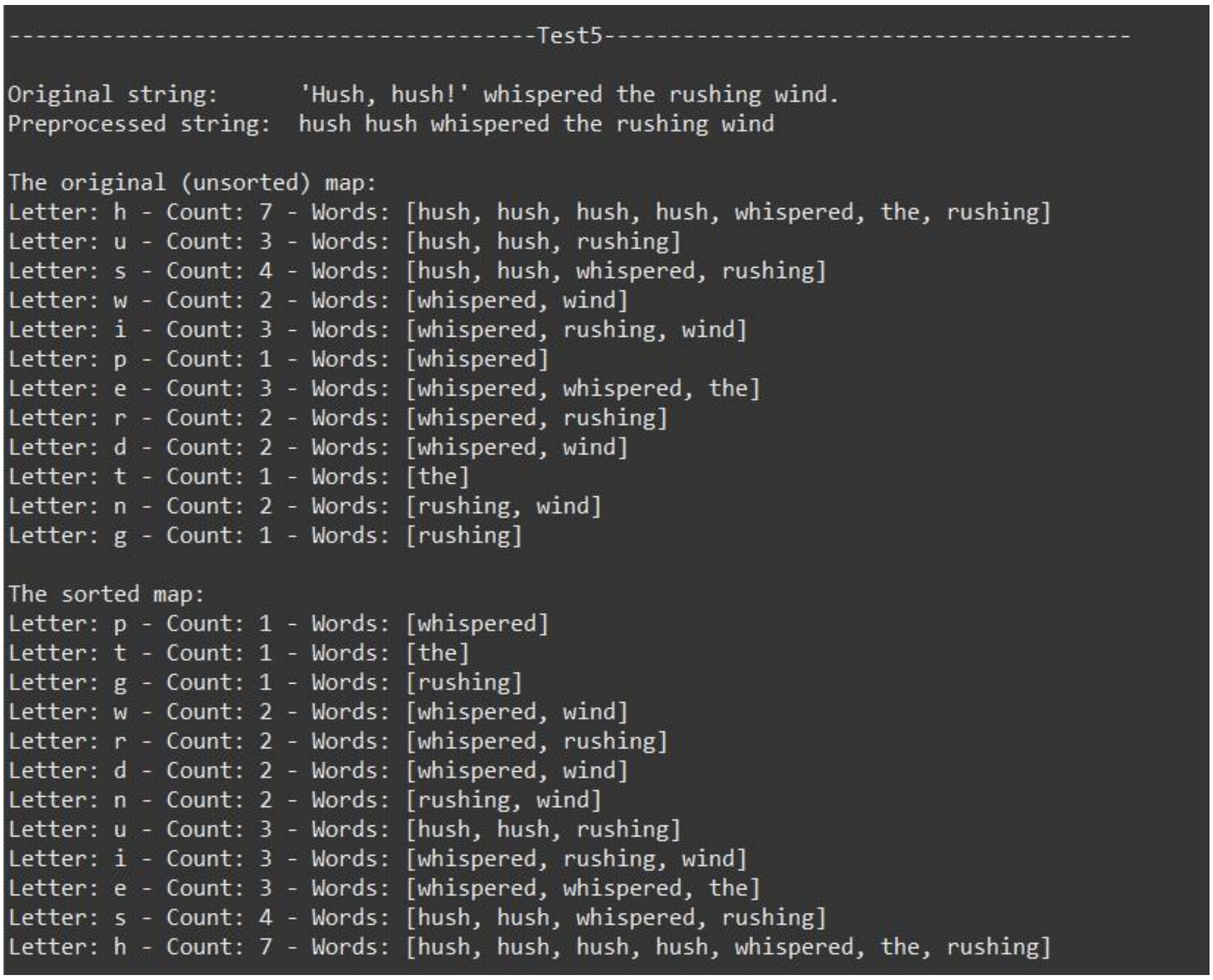
```
<terminated> Driver (1) [Java Application] C:\Users\Lenovo\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_
Hw6


----------------------------------------Test1----------------------------------------

Original string:      15//06//2023
!! Error. After the preprocessing step, there is no suitable string left !!

----------------------------------------Test2----------------------------------------

Original string:      'java'
Preprocessed string:  java

The original (unsorted) map:
Letter: j - Count: 1 - Words: [java]
Letter: a - Count: 2 - Words: [java, java]
Letter: v - Count: 1 - Words: [java]

The sorted map:
Letter: j - Count: 1 - Words: [java]
Letter: v - Count: 1 - Words: [java]
Letter: a - Count: 2 - Words: [java, java]
```



```
----------------------------------------Test3----------------------------------------

Original string:      abc aba
Preprocessed string:  abc aba

The original (unsorted) map:
Letter: a - Count: 3 - Words: [abc, aba, aba]
Letter: b - Count: 2 - Words: [abc, aba]
Letter: c - Count: 1 - Words: [abc]

The sorted map:
Letter: c - Count: 1 - Words: [abc]
Letter: b - Count: 2 - Words: [abc, aba]
Letter: a - Count: 3 - Words: [abc, aba, aba]
```



```
----------------------------------------Test4----------------------------------------

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]
```



```
---------------------------------------Test5---------------------------------------

Original string:      'Hush, hush!' whispered the rushing wind.
Preprocessed string:  hush hush whispered the rushing wind

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]
```