CSE222

HomeWork #6

Report

1901042676

Mustafa Mercan

Assignment:

This assignment consists of 3 parts.
In Part 1, using the myMap class we created, we are required to remove all characters in a string except letters and spaces. Then, we need to convert the remaining letters from uppercase to lowercase. Inside our myMap class, we have the following elements:

- LinkedHashMap<String, info> map
- int mapSize
- String str

In Part 2, after performing the above operation in Part 1, we are asked to store the resulting new string in a map object. This map structure has a key-value relationship. In our created map structure, the key represents a character, and the value represents an object called "info". The "info" object contains the following elements:

- int count
- String[] words

In Part 3, we are instructed to create a new class called "mergeSort".
In this class, we have the following variables: myMap originalMap, myMap sortedMap, and an optional variable called string[] aux. In this project, I chose not to use the "aux" variable. We are required to sort the "info" objects within the created map from Part 2 based on the "count" value using the merge sort algorithm.

Part1

In the first part of your assignment, you were asked to lowercase all the letters in a string and remove any characters except for spaces. Firstly, I printed the original string in the terminal. Then, I used the regex structure to perform this modification. Thanks to the regex structure, I were able to accomplish this task in a single line. Finally, you printed the modified string in the terminal.

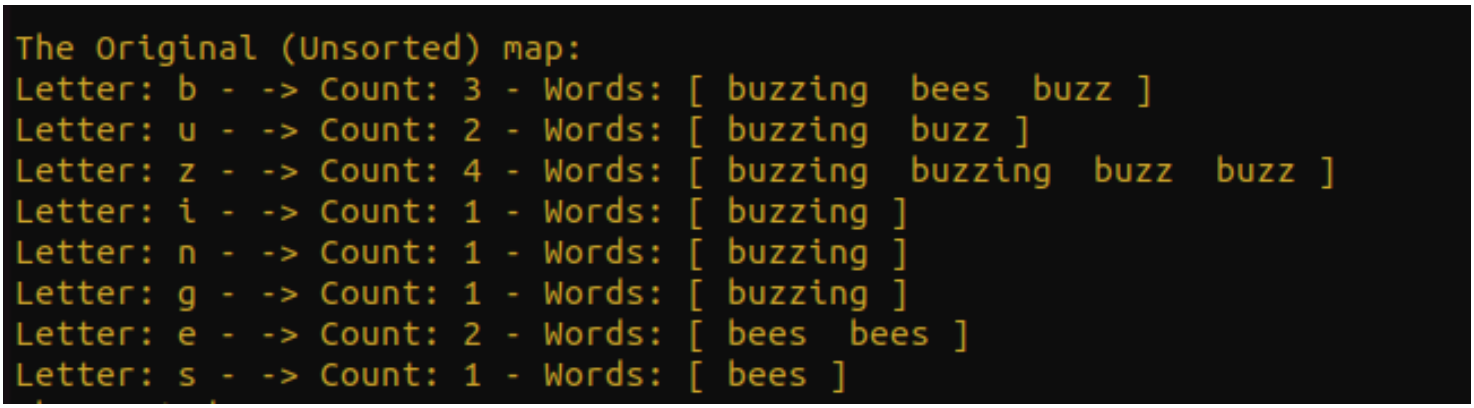
```
this.str = input.replaceAll("[^a-zA-Z\\s]", "").toLowerCase();
```

```
Original String -> Buzzing bees buzz.
Preprocessed String -> buzzing bees buzz
```

Part2

In Part 2 of the assignment, we were tasked with mapping the modifications we made to the string. To do this, I first parsed the string based on spaces. Then, I looped through the parsed string array. Within this loop, I used another loop to iterate through the letters of each word. If a letter was not already a key in our map, I created a new "info" object. Then, I used the add_word method I created within the "info" object to add the word to the ArrayList inside the "info" object. If the key already existed in the map, I accessed the corresponding value in the map using that key and added the word to the ArrayList inside the "info" object.

After performing all these operations, I printed the map to the terminal.



Part3

In Part 3 of the assignment, you were instructed to create a mergeSort class that takes the myMap object created in Part 2 as input. The task was to sort the map object based on the "count" variable inside the "info" object using the merge sort algorithm.

Firstly, for the merge sort algorithm, the list that holds our data should be divided into two halves. Then, these divided parts should be further divided into halves. This process should continue until there is only one element left in the list. Afterwards, the parts should be merged back together according to the desired condition. To achieve this, we should use a recursive function.

1→ First, we need to determine the return condition. For our merge sort algorithm, the return condition should be when there is only one element left, the return operation should be performed.

2→ Then, I created two myMap objects, leftMap and rightMap, which will be used to hold the divided parts of our input map.

3→ Next, I transferred all the keys belonging to our input map into an ArrayList. This way, we could iterate through our map.

4→With the help of the 'i' variable, I added the data to either the leftMap or the rightMap based on whether it is smaller than the 'middle' variable. This way, our input map was divided into two parts.

5→Then, I called the same function for leftMap first, and then for rightMap.

6→Then, I called a method named 'merge_map' to merge these two maps together and returned the result.

7→ Next, the first element in leftMap and rightMap is selected. Then, these elements are compared, and based on the correct condition, they are added to the mergeMap. The added element is then removed from either the leftMap or the rightMap. This process ends when one of the elements in rightMap or leftMap is finished.

8→ Finally, if there are remaining elements in either leftMap or rightMap, we add them to the mergeMap and return the mergeMap.

```
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 ]
```

To test the program, you can create a new object in the main file using the classes. The myMap object requires a constructor that takes a single string parameter, while the mergeSort object requires a constructor that takes a myMap object as an argument.

```
public static void main(String [] args)
{
    myMap test = new myMap("Buzzing bees buzz.");
    mergeSort test_2 = new mergeSort(test);
}
```

Error Cases

```
myMap test = new myMap("");
myMap test2 = new myMap("_*012%++'321^!");
myMap test3 = new myMap("                    ");


mergeSort ms1 = new mergeSort(test);
mergeSort ms2 = new mergeSort(test2);
mergeSort ms3 = new mergeSort(test3);
```

```
Original String ->
String is empty.

Original String -> _*012%++'321^!
String has any letter

Original String ->
String has any letter

Invalid input. Input is empty
Invalid input. Input is empty
Invalid input. Input is empty
```