GTU Department of Computer Engineering
CSE 222/505 - Spring 2023
Homework 6
Due date: May 13, 2023 – 23:59

In this homework you are supposed to use a custom Map structure and then sort it using Merge Sort.

Steps

1) *(10 points, no partial grading)* You take an input string. You can declare it within the code. Don't forget to print it to console. The input string is then preprocessed to make sure it contains only small lower-case letters. The preprocessing operation should perform the followings:
   - If there is a capital letter, convert it to lower-case.
   - If there is a character which is not a letter or space, delete it. Spaces should be kept in order to keep track of the words.

   Hint: The preprocessing can be done by using regex.

   Print the preprocessed string.

2) *(20 points, no partial grading)* Use the preprocessed string to build a custom map structure (named myMap). In this structure, you will basically keep the number of occurrences of each letter within the string along with the words, that it was seen. More information about this structure and examples are given below.

   Hint: If the string has X unique letters, then myMap should have X different keys, one for each.

   Print myMap.

3) *(70 points)* Write the merge sort code to sort myMap, in ascending order, according to the "count" values.

   Print sorted myMap.

Classes

1) Class "myMap":

You are supposed to write a class named myMap, to implement custom map structure. myMap should have the following variables:

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

So, basically, it uses LinkedHashMap with a String as key (Actually, the key is *Char* because it keeps a letter, but String will be easier to work on. Nevertheless, you are free to use *Char* instead of *String*.). But you have to use an object of "info" class as value.

2) Class "info":

As seen, the value of the LinkedHashMap is "info" which is another class you should implement with the following variables:

- *int* count
- *String* [] words

and a "void push" method to add another word to the "words" along with the increment operation of the counter.

To build myMap, you will work on each letter of each word of "str". For each letter, you should add it to the map as a key, if it wasn't added already. If the letter already exists in the map as a key, then you should update the value of it.

Example 1 of building a myMap object: Assume *str* is "java" (the string has only 1 word: *java*).

Then the "build map" operation should be done as following:

1. Add "j" to myMap with count value = 1 and words array = [java]
2. Add "a" to myMap with count value = 1 and words array = [java]
3. Add "v" to myMap with count value = 1 and words array = [java]
4. Update the value of key "a" as count = 2 and words array = [java, java] (since there are two a's in this word, you should add it twice).

The final version of the map will include the following:

- "j": count = 1, words = [java]
- "a": count = 2, words = [java, java]
- "v": count = 1, words = [java]

Example 2 of building a myMap object: Assume *str* is "abc aba" (the string has 2 words: *abc* and *aba*).

Then the "build map" operation should be done as following:

1. Add "a" to myMap with count value = 1 and words array = [abc]
2. Add "b" to myMap with count value = 1 and words array = [abc]
3. Add "c" to myMap with count value = 1 and words array = [abc]
4. Update the value of key "a" as count = 2 and words array = [abc, aba]
5. Update the value of key "b" as count = 2 and words array = [abc, aba]
6. Update the value of key "a" as count = 3 and words array = [abc, aba, aba]

The final version of the map will include the following:

- "a": count = 3, words = [abc, aba, aba]
- "b": count = 2, words = [abc, aba]
- "c": count = 1, words = [abc]

As seen, you have to traverse the whole string and make an operation (either add or update) for each letter. Your myMap class should have the necessary methods to perform given tasks.

3) Class "mergeSort"

In this class, you are going to implement merge sort algorithm. You should write the whole code by yourself, and the algorithm should be able to work with myMap structure (it doesn't have to work with simple arrays with integer values, you should modify the classic merge sort algorithm). This class should have the following variables:

- *myMap* originalMap
- *myMap* sortedMap (you are supposed to keep the sorted version of the map in a copy, do not edit the original map)
- *String []* aux

  "aux" can be a list or set, you are going to use this to keep track of sorted letters. Usage of this variable is not mandatory, but you cannot use any other auxiliary data type to make the sorting operation easier.

  Hint: You can use each key's count value to compare the entries in myMap, and save the related key (letter) into the aux array in the index it should be at. In other words, after sorting operation, the aux should be equal to the sorted version of keys. Also, the usage of aux is not mandatory, but it will make things easier.

Your mergeSort class should have the necessary methods to perform given tasks. You can implement a simple print method to print the map as seen in the examples below. If you need to add a new method to directly assign a myMap entry, you can add that method to the myMap class.

Examples

Example 1:

Input: "Buzzing bees buzz."

The input should be preprocessed and become: "buzzing bees buzz"

After that, the preprocessed string should be used to build a myMap as seen below (the tables are given for the easier understanding, you don't have to print a table):

Key (String)	Value (Info object)
b	Count: 3 - Words: [buzzing, bees, 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]

Finally, myMap object should be sorted according to the value's count variable.

Key (String)	Value (Info object)
i	Count: 1 - Words: [buzzing]
n	Count: 1 - Words: [buzzing]
g	Count: 1 - Words: [buzzing]
s	Count: 1 - Words: [bees]
u	Count: 2 - Words: [buzzing, buzz]
e	Count: 2 - Words: [bees, bees]
b	Count: 3 - Words: [buzzing, bees, buzz]
z	Count: 4 - Words: [buzzing, buzzing, buzz, buzz]

The console output should be as seen below:

```
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]
```

In example 1, the aux array will be [i, n, g, s, u, e, b, z] after the sorting operation (you don't have to print this information).

Example 2:

```
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]
```

In example 2, the aux array will be [p, t, g, w, r, d, n, u, i, e, s, h] after the sorting operation (you don't have to print this information).

PS: Be careful with the letters with the same frequency. In the sorted array, they should be listed in the input order (the order you have used to build myMap). In other words, if letter x and the letter y have the same frequency, the one that was seen first must be above the other. "Seen first" implies the first seen one among those two, when you process the string from the left end to the right end.

General Information

- You shouldn't use any libraries other than the ones below:
  - Java.util.List
  - Java.util.ArrayList
  - Java.util.Set
  - Java.util.Map
  - Java.util.LinkedHashMap

    PS: If you think a library other than the ones above is essential, contact us.

- Do not use any methods from the imported libraries to do the work you are asked to do. For instance, using a pre-defined method to implement merge sort causes getting 0 points from the related part.

Grading Details

No OOP Design	-100 points
No error handling	-50 points
No report (a report that explains the details and the manner of work of your solution)	-90 points
No javadoc documentation	-50 points
Cheating	-200 points
Using any library other than the ones stated above	-50 points
In step 3, partially sorting myMap.	-10 to - 60 points (according to the size of the unsorted part).
In step 3, sorting “count” values only, not myMap.	-60 points