GTU Department of Computer Engineering CSE 222/505 - Spring 2023

Homework 6 – Report

Erkut Dere – 1801042646

Code Structure:

public class homework6{

public static void main () {

//Get string input from user and check it

//Print original and preprocessed string

//Create a map and print unsorted map

//Create a sorted map from original map

}

public static class myMap() {

```
private LinkedHashMap<Character, info> mymap;
      private int mapSize;
      private String str;
public LinkedHashMap<Character, info> getMap()
public int getMapSize()
public String getStr()
public myMap(String str)
public String toString()
```

}

public static class info() {

```
private int count;
      private List<String> words;
public info(int count, List<String> words)
public void push(String word)
public int getCount()
```

```
public String toString()
}
public static class mergeSort() {
private LinkedHashMap<Character, info> originalMap;
      private LinkedHashMap<Character, info> sortedMap;
      private String[] aux;
public mergeSort(LinkedHashMap<Character, info> originalMap)
public void sort(LinkedHashMap<Character, info> map)
public void sort(LinkedHashMap<Character, info> map, int low, int high)
public void merge(LinkedHashMap<Character, info> map, int low, int mid, int high)
public void createSortedMap()
public String toString()
}
}
```

Sorting Algorithm:

In the first sort method (Wrapper method) I initialize the aux array because I will use it in the merge method to sort the map. After the initialization I call the real sort method.

```
public void sort(LinkedHashMap<Character, info> map) {
    for (int i = 0; i < this.originalMap.size(); i++) {
        this.aux[i] = map.keySet().toArray()[i].toString();
    }
    sort(map, low:0, map.size() - 1);
}
```

In this recursive sort method, while slicing the map into halves, the base case is the left index of the map must be smaller than the right index of the map so if (low < high) represents this situation, at every recursive call I calculated the mid index of the map and call the sort method for left and right halves of the map. In the last recursive call, we will start with the first merge method call by from left to right.

```java
public void sort(LinkedHashMap<Character, info> map, int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        sort(map, low, mid);
        sort(map, mid + 1, high);
        merge(map, low, mid, high);
    }
}
```

Merge method uses aux array to sort the original map. First, I created temporary arrays to store the data in the aux array (stored the characters in the aux array in the left and the right temporary string arrays). Then I compare the count value of characters in the original map (because aux array and the original map has the same index order) and if the left array's character's count value is smaller than the right array's character's count value, I put that character in that index otherwise I put the right array's character in the aux array in that index. After that I copied the remaining elements of both right and left arrays. At the end I will have the sorted aux array.

```java
public void merge(LinkedHashMap<Character, info> map, int low, int mid, int high) {

    // Find sizes of two subarrays to be merged
    int n1 = mid - low + 1;
    int n2 = high - mid;
    String[] left = new String[n1];
    String[] right = new String[n2];

    // Copy data to temp arrays
    for (int i = 0; i < n1; i++) {
        left[i] = this.aux[low + i];
    }
    for (int j = 0; j < n2; j++) {
        right[j] = this.aux[mid + 1 + j];
    }

    int i = 0, j = 0;
    int k = low;
    // Compare count of the letters in the map and sort them according to the count
    while (i < n1 && j < n2) {
        if (map.get(left[i].charAt(0)).count <= map.get(right[j].charAt(0)).count) {
            this.aux[k] = left[i];
            i++;
        } else {
            this.aux[k] = right[j];
            j++;
        }
        k++;
    }
    // Copy remaining elements of left array
    while (i < n1) {
        this.aux[k] = left[i];
        i++;
        k++;
    }
    // Copy remaining elements of right array
    while (j < n2) {
        this.aux[k] = right[j];
        j++;
        k++;
    }
}
```

In createSortedMap method I use the helper aux array to create sorted map object in mergeSort class, this process is the last part of the creating the sorted map:

```java
public void createSortedMap(){
    //Create sorted map by using aux array and get count and words from original map by using aux array
    for(int i = 0; i < this.aux.length; i++){
        this.sortedMap.put(this.aux[i].charAt(0), new info(this.originalMap.get(this.aux[i].charAt(0)).count, this.originalMap.get(this.aux[i].charAt(0)).words));
    }
    //print sorted map by using toString method in mergeSort class
    System.out.println(this.toString());
}
```

Some tests from my program:

Test1: When entering an empty string:

```
erkut@DESKTOP-7JISF9F:~/hw6$ javac homework6.java
erkut@DESKTOP-7JISF9F:~/hw6$ java homework6
Enter a string:
You entered an empty string!
```

Test2: When entering a non-letter characters:

```
erkut@DESKTOP-7JISF9F:~/hw6$ javac homework6.java
erkut@DESKTOP-7JISF9F:~/hw6$ java homework6
Enter a string: '^#
You entered a string that only contains non letter charachters!
erkut@DESKTOP-7JISF9F:~/hw6$
```

```
erkut@DESKTOP-7JISF9F:~/hw6$ java homework6
Enter a string: 32
You entered a string that only contains non letter charachters!
erkut@DESKTOP-7JISF9F:~/hw6$
```

Some outputs when user enters correct input strings:

```
erkut@DESKTOP-7JISF9F:~/hw6$ java homework6
Enter a string: 32java32


Original string:        32java32
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]
```

```
erkut@DESKTOP-7JISF9F:~/hw6$ java homework6
Enter a string: Buzzing bees buzz.


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]

erkut@DESKTOP-7JISF9F:~/hw6$
```

```
erkut@DESKTOP-7JISF9F:~/hw6$ java homework6
Enter a string: 'Hush, hush!' whispered the rushing wind.


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]

erkut@DESKTOP-7JISF9F:~/hw6$
```