GTU Department Of Computer Engineering

CSE 222 / 505 – Spring 2023

Homework 6

Ahmet CEBECİ

1901042708

CODE FUNCTIONALITY

1 – class “info”

```java
/**
 * info class
 *
 */
public class info {
    int count;
    String[] words;

    /**
     * Default constructor
     *
     */
    public info() {
        count = 0;
        words = new String[0];
    }
    /**
     *
     * @param word
     */
    public void push(String word) {
        String[] newWords = new String[words.length + 1];
        System.arraycopy(words, 0, newWords, 0, words.length);
        newWords[words.length] = word;
        words = newWords;
        count++;
    }
```

This info class is a utility class that's designed to keep track of certain pieces of information about a character in a string.

It has two data members: count and words. The count member is an integer that tracks how many times a character has been encountered. The words member is an array of

strings that keeps track of the words in which the character appears.

The info class has a default constructor which initializes count to 0 and words to an empty string array.

The push method is used to add a word to the words array and increment the count. When push is called with a word as an argument, it creates a new array that's one element larger than the current words array. It then copies the existing words into the new array, adds the new word at the end, and replaces the old words array with the new one. It also increments count by one.

2 – class “myMap”

```java
public class myMap {
    LinkedHashMap<String, info> map;
    int mapSize;
    String str;

    /**
     * Default constructor
     * @param str
     */
    public myMap(String str) {
        this.str = str;
        map = new LinkedHashMap<>();
        mapSize = 0;
        String[] words = str.split(regex:" ");
        for (String word : words) {
            for (char letter : word.toCharArray()) {
                String letterStr = String.valueOf(letter);
                map.putIfAbsent(letterStr, new info());
                map.get(letterStr).push(word);
            }
        }
        mapSize = map.size();
    }
    /**
     *
     */
    protected void printMap() {
        for (String key : map.keySet()) {
            StringBuilder sb = new StringBuilder();
            sb.append(str:"[");
            for (int i = 0; i < map.get(key).words.length; i++) {
                sb.append(map.get(key).words[i]);
                if (i < map.get(key).words.length - 1) {
                    sb.append(str:", ");
                }
            }
            sb.append(str:"]");
            System.out.println("Letter: " + key + " - Count: " + map.get(key).count + " - Words: " + sb.toString());
        }
    }
}
```

This class, myMap, is designed to process a string and create a map where each character from the string is a key linked to information about that character. The information includes the number of occurrences of the character and the words in which the character appears.

When you instantiate a myMap with a string, the constructor splits the string into words and then processes each word character by character. For each character, it checks whether it's already in the map. If not, it adds the character as a key with a new info object as its value. If the character is already

in the map, it updates the associated info object with the current word.

The printMap() method can then be used to display the data stored in the map. It iterates over each key-value pair in the map, and for each character, it prints out the character, the count of its occurrences, and a list of words in which it appears.

3 – class “mergeSort”

```java
 */
public mergeSort(myMap map) {
    originalMap = map;
    sortedMap = new myMap(map.str);
    aux = new ArrayList<>();
    aux.addAll(originalMap.map.keySet());
    mergeSortFunc(low:0, aux.size() - 1);
    LinkedHashMap<String, info> sorted = new LinkedHashMap<>();
    for (String key : aux) {
        sorted.put(key, originalMap.map.get(key));
    }
    sortedMap.map = sorted;
}
/**
 *
 * @param low
 * @param high
 */
private void mergeSortFunc(int low, int high) {
    if (high <= low) {
        return;
    }

    int mid = (low + high)/2;
    mergeSortFunc(low, mid);
    mergeSortFunc(mid + 1, high);

    ArrayList<String> left = new ArrayList<>(aux.subList(low, mid + 1));
    ArrayList<String> right = new ArrayList<>(aux.subList(mid + 1, high + 1));

    int i = 0;
    int j = 0;
    for (int k = low; k <= high; k++) {
        if (i >= left.size()) {
            aux.set(k, right.get(j++));
        }
        else if (j >= right.size() || originalMap.map.get(left.get(i)).count <= originalMap.map.get(right.get(j)).count) {
            aux.set(k, left.get(i++));
        }
        else {
            aux.set(k, right.get(j++));
        }
    }
}
```

```java
protected void printSortedMap() {
    for (String key : sortedMap.map.keySet()) {
        StringBuilder sb = new StringBuilder();
        sb.append(str:"[");
        for (int i = 0; i < sortedMap.map.get(key).words.length; i++) {
            sb.append(sortedMap.map.get(key).words[i]);
            if (i < sortedMap.map.get(key).words.length - 1) {
                sb.append(str:", ");
            }
        }
        sb.append(str:"]");
        System.out.println("Letter: " + key + " - Count: " + sortedMap.map.get(key).count + " - Words: " + sb.toString());
    }
}
```

The mergeSort class is designed to sort the keys of a myMap instance according to the count of their occurrences in a given string. It utilizes the merge sort algorithm, which is a comparison-based sorting algorithm that operates in a divide-and-conquer fashion.

Upon instantiation with a myMap object, the mergeSort constructor creates a copy of the original map and a list (aux) of the keys of the original map. The keys in the aux list are then sorted with the mergeSortFunc method.

The mergeSortFunc method recursively divides the list of keys into two halves until it has pairs of keys to compare. It then merges these pairs back together in sorted order, according to the counts of their occurrences.

Once the keys are sorted, a new LinkedHashMap sorted is created, preserving the order of the keys from the aux list. The sorted map is then assigned to the sortedMap's map.

The printSortedMap method is used to print the data stored in the sortedMap. It iterates over each key-value pair in the sortedMap, and for each character, it prints out the character, the count of its occurrences, and a list of words in which it appears. This allows the user to see the characters sorted by their frequency of appearance in the string.

4 – class “mainTest”

```java
public static void main(String[] args) {
    run();
}
/**
 *
 */
public static void run() {
    // Example 1
    System.out.println(x:"Example 1:");
    try {
        String input1 = "";
        String preprocessed1 = preprocess(input1);
        System.out.println("Original String: " + input1);
        System.out.println("Preprocessed String: " + preprocessed1);


        myMap myMap1 = new myMap(preprocessed1);
        System.out.println(x:"\nThe original (unsorted) map:");
        myMap1.printMap();

        mergeSort mergeSort1 = new mergeSort(myMap1);
        System.out.println(x:"\nThe sorted map:");
        mergeSort1.printSortedMap();

    }
    catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
    }

    // Example 2
    System.out.println(x:"\nExample 2:");
    try {
        String input2 = "Buzzing bees buzz.";
        String preprocessed2 = preprocess(input2);
        System.out.println("Original String: " + input2);
        System.out.println("Preprocessed String: " + preprocessed2);

        myMap myMap2 = new myMap(preprocessed2);
        System.out.println(x:"\nThe original (unsorted) map:");
        myMap2.printMap();

        mergeSort mergeSort2 = new mergeSort(myMap2);
        System.out.println(x:"\nThe sorted map:");
        mergeSort2.printSortedMap();
    }
    catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
    }
```

```
        // Example 3
        System.out.println(x:"\nExample 3:");
        try {
            String input3 = "'Hush, hush!' whispered the rushing wind.";
            String preprocessed3 = preprocess(input3);
            System.out.println("Original String: " + input3);
            System.out.println("Preprocessed String: " + preprocessed3);

            myMap myMap3 = new myMap(preprocessed3);
            System.out.println(x:"\nThe original (unsorted) map:");
            myMap3.printMap();

            mergeSort mergeSort3 = new mergeSort(myMap3);
            System.out.println(x:"\nThe sorted map:");
            mergeSort3.printSortedMap();
        }
        catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
    /**
     *
     * @param input
     * @return
     */
    private static String preprocess(String input) {
        // if input is null, return null
        if (input.equals(anObject:"")) {
            System.out.println(x:"Input is null.");
            return null;
        }
        String preprocessed = input.toLowerCase();
        preprocessed = preprocessed.replaceAll(regex:"[^a-z ]", replacement:"");
        return preprocessed;
    }

}
```

This Java program preprocessed text inputs, creates a map from the preprocessed text, and sorts the map. It handles exceptions by printing an error message and continuing with the next example. The preprocessing step converts the text to lowercase and removes non-alphabetical characters, returning null for empty inputs. The mapping and sorting are handled by the myMap and mergeSort classes respectively.

OUTPUT

```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/1901042708_Ahmet_Cebeci_CSE222_HW6$ cd hw6
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/1901042708_Ahmet_Cebeci_CSE222_HW6/hw6$ javac *.java
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/1901042708_Ahmet_Cebeci_CSE222_HW6/hw6$ cd ..
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/1901042708_Ahmet_Cebeci_CSE222_HW6$ java hw6.mainTest
```

```
Example 1:
Input is null.
Original String:
Preprocessed String: null
Exception: null
```

```
Example 2:
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]
```

```
Example 3:
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]
```