Hw6 report CSE222 Eren Torlak 210104004090

Step 1: Preprocessing the input string

[^a-zA-Z\\s] : ^ means not, a-zA-Z means all letters, \\s means space

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

Step 2: Build the map

buildMap function :

1. split the string into words
2. for each word, get the characters
3. for each character, if the character is not in the map, add it to the map and increment the count by 1. if the character is in the map, increment the count by 1.
4. add the word to the words arraylist

Step 3:

```
    public void sortMap() {
        ArrayList<String> keys = new
ArrayList<>(originalMap.getMap().keySet());
        mergeSortHelper(keys, 0, keys.size() - 1);
        for (String key : keys) {
            sortedMap.getMap().put(key, originalMap.getMap().get(key)); // add
the sorted keys to the sorted map
        }
    }
```

The sortMap():

1. The method first creates an ArrayList of the keys in the original map.
2. The method then calls the mergeSortHelper() method to sort the ArrayList.
3. The method then iterates over the sorted ArrayList and adds the keys to a new map.
4. The method returns the new map.

Hw6 report CSE222 Eren Torlak 210104004090

The mergeSortHelper():

The mergeSortHelper() method is a recursive algorithm that sorts an ArrayList of keys by their counts. The algorithm works by first dividing the ArrayList into two halves. It then recursively sorts the two halves. Finally, it merges the two sorted halves back into the original ArrayList.

The merge() algorithm:

1. The algorithm first initializes two indices, leftiter and rightiter. The index leftiter points to the beginning of the left half of the array, and the index rightiter points to the beginning of the right half of the array.
2. The algorithm then initializes a third index, auxiter. The index auxiter points to the beginning of an auxiliary array.
3. The algorithm then iterates over the left and right halves of the array. For each iteration, the algorithm compares the counts of the keys at the current indices of the left and right halves.
4. If the count of the key at index leftiter is less than or equal to the count of the key at index rightiter, then the algorithm adds the key at index leftiter to the auxiliary array. The algorithm then increments the index leftiter by 1.
5. If the count of the key at index rightiter is less than the count of the key at index leftiter, then the algorithm adds the key at index rightiter to the auxiliary array. The algorithm then increments the index rightiter by 1.
6. The algorithm continues iterating until it reaches the end of one of the halves of the array.
7. Once the algorithm has reached the end of one of the halves of the array, it adds the remaining keys from the other half to the auxiliary array.
8. The algorithm then copies the sorted contents of the auxiliary array back to the original array.
9. The algorithm returns.
10. N(logN)

Hw6 report CSE222 Eren Torlak 210104004090

```java
/**
     * This method merges the two halves of the array
     *
     * The idea is to compare the counts of the two keys and add the smaller one to
     * the aux array
     * if the counts are equal, add the key that comes first in the original string
     *
     * @param keys
     * @param left
     * @param mid
     * @param right
     */
    private void merge(ArrayList<String> keys, int left, int mid, int right) {
        int leftIter = left;
        int rightIter = mid + 1;

        int auxIter = left;

        // compare the counts of the two keys and add the smaller one to the aux array
        // if the counts are equal, add the key that comes first in the original string
        String keyLeft, keyRight;
        while (leftIter <= mid && rightIter <= right) {
            keyLeft = keys.get(leftIter);
            keyRight = keys.get(rightIter);

            // if the counts are equal, left one comes first
            if (originalMap.getMap().get(keyLeft).getCount() <= originalMap.getMap().get(keyRight).getCount()) {
                aux[auxIter] = keyLeft; // add the key to the aux array and increment auxIter for the next key
                auxIter++;
                leftIter++; // increment leftIter for the loop to continue
            } else {
                aux[auxIter] = keyRight;
                auxIter++;
                rightIter++; // increment rightIter for the loop to continue
            }
        }
        // add the remaining keys. left to mid
        while (leftIter <= mid) {
            aux[auxIter] = keys.get(leftIter);
            auxIter++;
```

```
51.             leftIter++;
52.         }
53.         // add the remaining keys .mid to right
54.         while (rightIter <= right) {
55.             aux[auxIter] = keys.get(rightIter);
56.             auxIter++;
57.             rightIter++;
58.         }
59.         // copy the sorted aux array back to the original array
60.         for (int iter = left; iter <= right; iter++) {
61.             keys.set(iter, aux[iter]);
62.         }
63.     }
64.
65.     /**
66.      * This method prints the original map
67.      */
68.     public void printOriginalMap() {
69.         System.out.println("The original (unsorted) map:");
70.         originalMap.printMap();
71.     }
72.
73.     /**
74.      * This method prints the original map and the sorted map
75.      */
76.     public void printSortedMap() {
77.         System.out.println("The sorted map:");
78.         sortedMap.printMap();
79.     }
```