Solution Review: Problem Challenge 1

We'll cover the following

Rearrange String K Distance Apart (hard)

Given a string and a number 'K', find if the string can be rearranged such that the same characters are at least 'K' distance apart from each other.

Example 1:

```
Input: "mmpp", K=2
Output: "mpmp" or "pmpm"
Explanation: All same characters are 2 distance apart.
```

Example 2:

```
Input: "Programming", K=3
Output: "rgmPrgmiano" or "gmringmrPoa" or "gmrPagimnor" and a few more
Explanation: All same characters are 3 distance apart.
```

Example 3:

```
Input: "aab", K=2
Output: "aba"
Explanation: All same characters are 2 distance apart.
```

Example 4:

```
Input: "aappa", K=3
Output: ""
Explanation: We cannot find an arrangement of the string where any two 'a' are 3 distance apar
t.
```

Solution

This problem follows the Top 'K' Numbers pattern and is quite similar to Rearrange String. The only difference is that in the 'Rearrange String' the same characters need not be adjacent i.e., they should be at least '2' distance apart (in other words, there should be at least one character between two same characters), while in the current problem, the same characters should be 'K' distance apart.

Following a similar approach, since we were inserting a character back in the heap in the next iteration, in this problem, we will re-insert the character after 'K' iterations. We can keep track of previous characters in a queue to insert them back in the heap after 'K' iterations.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```java
import java.util.*;

class RearrangeStringKDistanceApart {

  public static String reorganizeString(String str, int k) {
    if (k <= 1)
      return str;

    Map<Character, Integer> charFrequencyMap = new HashMap<>();
    for (char chr : str.toCharArray())
      charFrequencyMap.put(chr, charFrequencyMap.getOrDefault(chr, 0) + 1);

```

```
    PriorityQueue<Map.Entry<Character, Integer>> maxHeap = new PriorityQueue<Map.Entry<Character, Integer
        (e1, e2) -> e2.getValue() - e1.getValue());

    // add all characters to the max heap
    maxHeap.addAll(charFrequencyMap.entrySet());

    Queue<Map.Entry<Character, Integer>> queue = new LinkedList<>();
    StringBuilder resultString = new StringBuilder(str.length());
    while (!maxHeap.isEmpty()) {
      Map.Entry<Character, Integer> currentEntry = maxHeap.poll();
      // append the current character to the result string and decrement its count
      resultString.append(currentEntry.getKey());
      currentEntry.setValue(currentEntry.getValue() - 1);
      queue.offer(currentEntry);
      if (queue.size() == k) {
        Map.Entry<Character, Integer> entry = queue.poll();
```

Run Save Reset

Time complexity

The time complexity of the above algorithm is $O(N * logN)$ where 'N' is the number of characters in the input string.

Space complexity

The space complexity will be $O(N)$, as in the worst case, we need to store all the 'N' characters in the HashMap.

Back

Problem Challenge 1

Next

Problem Challenge 2

Mark as Completed

Report an Issue Ask a Question