Frequency Sort (medium)

We'll cover the following

Problem Statement

Given a string, sort it based on the decreasing frequency of its characters.

Example 1:

```
Input: "Programming"
Output: "rrggmmPiano"
Explanation: 'r', 'g', and 'm' appeared twice, so they need to appear before any other characte
r.
```

Example 2:

```
Input: "abcbab"
Output: "bbbaac"
Explanation: 'b' appeared three times, 'a' appeared twice, and 'c' appeared only once.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
const sort_character_by_frequency = function(str) {
  // TODO: Write your code here
  return "";
};


console.log(`String after sorting characters by frequency: ${sort_character_by_frequency("Programming")}`)
console.log(`String after sorting characters by frequency: ${sort_character_by_frequency("abcbab")}`)

```

RUN SAVE RESET

Solution

This problem follows the **Top 'K' Elements** pattern, and shares similarities with Top 'K' Frequent Numbers.

We can follow the same approach as discussed in the Top 'K' Frequent Numbers problem. First, we will find the frequencies of all characters, then use a max-heap to find the most occurring characters.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
const Heap = require('./collections/heap'); //http://www.collectionsjs.com


function sort_character_by_frequency(str) {
  // find the frequency of each character
  charFrequencyMap = {};
  for (i = 0; i < str.length; i++) {
    const chr = str[i];
    if (chr in charFrequencyMap) {
      charFrequencyMap[chr]++;
    } else {
      charFrequencyMap[chr] = 1;
    }
  }

  // add all characters to the max heap
  const maxHeap = new Heap([], null, ((a, b) => a[0] - b[0]));
  Object.keys(charFrequencyMap).forEach((key) => {
    maxHeap.push([charFrequencyMap[key], key]);
  });

  // build a string, appending the most occurring characters first
  const sortedString = [];
  while (maxHeap.length > 0) {
    [frequency, char] = maxHeap.pop();
    for (let i = 0; i < frequency; i++) {
```

```
27    sortedString.push(char);
28  }
```

RUN SAVE RESET

Close

Output 3.823s

```
String after sorting characters by frequency: rrggmmoaPin
String after sorting characters by frequency: bbbaac
```

Time complexity

The time complexity of the above algorithm is $O(D * logD)$ where 'D' is the number of distinct characters in the input string. This means, in the worst case, when all characters are unique the time complexity of the algorithm will be $O(N * logN)$ where 'N' is the total number of characters in the 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
Top 'K' Frequent Numbers (medium)

Next
Kth Largest Number in a Stream (medi...

Mark as Completed

Report an Issue Ask a Question