Frequency Sort (medium)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
  - Time complexity
  - Space complexity

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++

```python
def sort_character_by_frequency(str):
  # TODO: Write your code here
  return ""


def main():

  print("String after sorting characters by frequency: " +
        sort_character_by_frequency("Programming"))
  print("String after sorting characters by frequency: " +
        sort_character_by_frequency("abcbab"))


main()

```

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

```python
from heapq import *


def sort_character_by_frequency(str):

  # find the frequency of each character
  charFrequencyMap = {}
  for char in str:
    charFrequencyMap[char] = charFrequencyMap.get(char, 0) + 1

  maxHeap = []
  # add all characters to the max heap
  for char, frequency in charFrequencyMap.items():
    heappush(maxHeap, (-frequency, char))

  # build a string, appending the most occurring characters first
  sortedString = []
  while maxHeap:
    frequency, char = heappop(maxHeap)
    for _ in range(-frequency):
      sortedString.append(char)

  return ''.join(sortedString)


def main():

  print("String after sorting characters by frequency: " +
        sort_character_by_frequency("Programming"))
  print("String after sorting characters by frequency: " +
        sort_character_by_frequency("abcbab"))


main()

```

Run | Save | Reset

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...

Completed

Report an Issue | Ask a Question