423. Reconstruct Original Digits from English

Feb. 10, 2019 | 2.2K views

Given a **non-empty** string containing an out-of-order English representation of digits `0-9`, output the digits in ascending order.

Note:

1. Input contains only lowercase English letters.
2. Input is guaranteed to be valid and can be transformed to its original digits. That means invalid inputs such as "abc" or "zerone" are not permitted.
3. Input length is less than 50,000.

Example 1:

```
Input: "owoztneoer"

Output: "012"
```

Example 2:

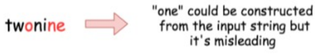
```
Input: "fviefuro"

Output: "45"
```

Solution

Approach 1: Hashmap

Intuition

The naive approach would be to construct as many `"zero"`s as it's possible from letters available in the input string, then as many `"one"`s as it's possible, etc. The problem is that letters `"o"`, `"n"`, `"e"` could be present as well in the another numbers that means that the straightforward approach could be misleading.



Hence the idea is to look for something unique. One could notice that all even numbers contain each a unique letter :

- Letter "z" is present only in "zero".
- Letter "w" is present only in "two".
- Letter "u" is present only in "four".
- Letter "x" is present only in "six".
- Letter "g" is present only in "eight".

> Hence there is a good way to count even numbers.

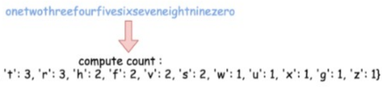
That is actually the key how to count `3`s, `5`s and `7`s since some letters are present only in one odd and one even number (and all even numbers has already been counted) :

- Letter "h" is present only in "three" and "eight".
- Letter "f" is present only in "five" and "four".
- Letter "s" is present only in "seven" and "six".

Now one needs to count `9`s and `1`s only, and the logic is basically the same :

- Letter "i" is present in "nine", "five", "six", and "eight".
- Letter "n" is present in "one", "seven", and "nine".

Implementation



Java Python Copy

```python
class Solution:
    def originalDigits(self, s: 'str') -> 'str':
        # building hashmap letter -> its frequency
        count = collections.Counter(s)

        # building hashmap digit -> its frequency
        out = {}
        # letter "z" is present only in "zero"
        out["0"] = count["z"]
        # letter "w" is present only in "two"
        out["2"] = count["w"]
        # letter "u" is present only in "four"
        out["4"] = count["u"]
        # letter "x" is present only in "six"
        out["6"] = count["x"]
        # letter "g" is present only in "eight"
        out["8"] = count["g"]
        # letter "h" is present only in "three" and "eight"
        out["3"] = count["h"] - out["8"]
        # letter "f" is present only in "five" and "four"
        out["5"] = count["f"] - out["4"]
        # letter "s" is present only in "seven" and "six"
        out["7"] = count["s"] - out["6"]
        # letter "i" is present in "nine", "five", "six", and "eight"
        out["9"] = count["i"] - out["5"] - out["6"] - out["8"]
        # letter "n" is present in "one", "nine", and "seven"
        out["1"] = count["n"] - out["7"] - 2 * out["9"]
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ where `N` is a number of characters in the input string. $\mathcal{O}(N)$ time is needed to compute hashmap `count` "letter -> its frequency in the input string". Then we deal with a data structure `out` which contains `10` elements only and all operations are done in a constant time.
- Space complexity : $\mathcal{O}(1)$. `count` contains constant number of elements since input string contains only lowercase English letters, and `out` contains not more than `10` elements.

Rate this article:

Previous Next

Comments: 1 Sort By

Type comment here... (Markdown is supported)

Preview Post

vivekchand19 4 April 18, 2019 6:27 PM

Typo: Should be `Now one needs to count 9s and 1s only, and the logic is basically the same :`

1 Share Reply

SHOW 1 REPLY