LeetCode Explore Problems Interview Contest Discuss Store Premium

Description Solution Discuss (487) Submissions

Back

Python 3 solution - with process of thinking and improvement

dmitrasulev ★ 303 Last Edit: February 14, 2021 4:15 AM 954 VIEWS

42

Part 1

Sunday morning, Spotify, coffee, console... Task from the list of tasks to solve later.
Let's go!

```
~ ) python3                                                                    at 10:38:03
Python 3.9.1 (default, Feb  3 2021, 07:38:02)
[Clang 12.0.0 (clang-1200.0.32.29)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> nums = [1,1,2,2,2,6,6,4,4,5,5,3]
```

Ok, so I need to count frequency of each of the unique values... First idea - use `Counter`. This returns `collection` type of object:

```
>>> d = Counter(nums)
>>> d
Counter({2: 3, 1: 2, 6: 2, 4: 2, 5: 2, 3: 1})
>>> type(d)
<class 'collections.Counter'>
```

Now I need to sort those values following the requirements in the description. 'Counter' object has no attribute 'sort', and 'sorted' will only sort values, not frequencies. Googling options, found this StackOverflow question with lots of useful answers. Reading... Let's try easier object:

```
>>> r = Counter(nums).most_common()
```

This returns a list of tuples, where first number is value and second - its' frequency. Can I sort it in the same command? Jumping to the official documentation. Nope, not sortable in the command itself, moreover: "*Elements with equal counts are ordered in the order first encountered*". Ok, let's sort it directly, first by values in the decreasing order, then by frequencies in the increasing.

```
>>> r.sort(key = lambda x: x[0], reverse=True)
>>> r.sort(key = lambda x: x[1])
>>> r
[(3, 1), (6, 2), (5, 2), (4, 2), (1, 2), (2, 3)]
```

Looks promising. Now I want to expand those tuples into a single list... Still browsing answers to the same question. Remembering that I can expand tuple and get every number from it by using this:

```
>>> a, b = (3, 2)
>>> a
3
>>> b
2
```

so then I can repeat every value by the number of its' frequency like so:

```
>>> [3]*2
[3, 3]
```

Aha. Now I need an empty list to combine all those tuples into a single list:

```
t = []
for i in r:
    a, b = i
    t.extend([a] * b)

>>> t
[3, 6, 6, 5, 5, 4, 4, 1, 1, 2, 2, 2]
```

Woo-hoo! That's what I need. So the complete solution now looks like this:

```
class Solution:
    def frequencySort(self, nums: List[int]) -> List[int]:

        r = Counter(nums).most_common()
        r.sort(key = lambda x: x[0], reverse=True)
        r.sort(key = lambda x: x[1])

        t = []
        for i in r:
            a, b = i
            t.extend([a]*b)

        return t
```

Result:
Runtime: 52 ms, faster than 63.30% of Python3 online submissions for Sort Array by Increasing Frequency.
Memory Usage: 14.2 MB, less than 58.20% of Python3 online submissions for Sort Array by Increasing Frequency.

Not the best, but the task is solved.

Part 2

Now it's time for another *fun* - can I make it one-liner or optimize the solution in any other way?

Looking at sorting lines... Can I sort it in one go? Yes! So, first we sort by values in the reverse order (`-x[0]`) and then by their frequencies (`x[1]`) in direct order.

```
>>> r.sort(key = lambda x: (x[1], -x[0]))
```

Basically, it's the same operation as the above but now coded in one line. Love Python :) Same logic applies to the tuple expansion part and it allows to save another line:

```
t = []
for i in r:
    t += ([i[0]] * i[1])
```

And then I thought - if I can sort by value and its' frequency why do I need intermediate list? Can I sort the original list the same way?! Let's see...

```
>>> nums
[1, 1, 2, 2, 2, 6, 6, 4, 4, 5, 5, 3]
>>> r = Counter(nums)
>>> r
Counter({2: 3, 1: 2, 6: 2, 4: 2, 5: 2, 3: 1})
>>> nums.sort(key=lambda x: (r[x], -x))
>>> nums
[3, 6, 6, 5, 5, 4, 4, 1, 1, 2, 2, 2]
```

Voila! That feels sooo good. But `x.sort` makes it in-place and I need to return an object... So, I need to change it to `sorted` then:

```
>>> result = sorted(nums, key=lambda x: (r[x], -x))
>>> result
[3, 6, 6, 5, 5, 4, 4, 1, 1, 2, 2, 2]
```

Perfect. So the final variant would be:

```
class Solution:
    def frequencySort(self, nums: List[int]) -> List[int]:
        r = Counter(nums)
        return sorted(nums, key=lambda x: (r[x], -x))
```

And it's even faster!

Runtime: 44 ms, faster than 95.07% of Python3 online submissions for Sort Array by Increasing Frequency.
Memory Usage: 14.3 MB, less than 58.20% of Python3 online submissions for Sort Array by Increasing Frequency.

NOTE

- If you want to downvote this post, please, be brave and comment why. This will help me and others to learn from it.
- If you think that others can learn something from this post, then please, upvote it. Thanks.

python thinking process python 3 thinking

Comments: 5 Best Most Votes Newest to Oldest Oldest to Newest